COMPUTER - AIDED ANALYSIS OF BUSINESS DATA PROCESSING SYSTEMS

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of
DOCTOR OF PHILOSOPHY

By
ASHA GOYAL

to the
COMPUTER SCIENCE PROGRAM
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR
MARCH, 1978

CSP-1970-D-GOY-COM

to
my PARENTS
and
THOSE, who have been close

CERTIFICATE

THIS is to certify that the work entitled "COMPUTER-AIDED ANALYSIS OF BUSINESS DATA PROCESSING SYSTEMS" by Asha Goyal has been carried out under my supervision and has not been submitted elsewhere for a degree.

Kanpur
March 1978

V. Rajaraman
Professor
Computer Science Program
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

8.9.1979

ACKNOWLEDGEMENTS

I am deeply indebted to my thesis advisor, Professor V. Rajaraman, who has always been a source of confidence. His encouragement; advice and interest in the work has been invaluable toward the completion of this thesis. His guidance is acknowledged with gratitude.

I am thankful to all my colleagues for many useful discussions, I have had with them. I wish to thank Dr. M.S. Krishnamoorthy, Dr. A.S. Sethi, Dr. B.K. Gairola and the friends who have often provided a much needed atmosphere to carry on the work of research.

I wish to express my thanks to Mr. H.K. Nathani who has taken pains to do the difficult job of typing the manuscript.

Kanpur
March 1978

- Asha Goyal

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Synopsis

of the

Ph.D. Dissertation
on

COMPUTER-AIDED ANALYSIS OF BUSINESS DATA PROCESSING SYSTEMS

BY

Asha Goyal
Computer Science Programme
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

March 1978

Information system design for administrative systems has always taken enormous time and manpower. Most often, the reason for this is a communication gap between the user and the systems analyst created by an inadequate description of the user's problem. This calls for many iterations of the various phases of the system development cycle. The problem of reducing the time and efforts required for the information system development has been taken up in this thesis. The main idea is to introduce the computer as a tool in the system development cycle.

A lot of work is being done in this direction. Some of the systemslike Hoskyn's System, ISDOS and PROTO SYSTEM I - BOTTOM PART etc., have tried to use the computer as an aid for the automation of the design phase. Systems are also being developed to automate the problem acquisition phase. The business definition language attempts to describe the functions of the organisation.

Query-by-example, SEQUEL, RIL and HI-IQ attempt to use tuple based data selection for the problem description.

In this thesis we approach the problem by considering system description and system analysis phases as of basic importance because of the dependence of other phases on these two phases. To make the approach practicable, the problem acquisition phase is reduced from a function description of the user's problem to the description of user's data processing requirements only.

A language is needed for data processing systems which will be able to describe the data transformations needed by the user without getting into the procedural details of how such data would be managed. This language should also be suitable for precisely describing the interaction of the target information system with its environment. The basic specification of an information system can be given by describing its inputs, outputs or reports, associated timings and the data transformations.

The proposed methodology considers reports as the starting point of the information system description. A language is developed for describing the reports completely by giving the logical structure of the reports. Special features are incorporated in this language to describe aspects related to the reporting medium or the device. The nature of input documents is similar to that

of reports. Thus this language also has features to describe input documents.

The second part of this language incorporates features needed for describing data association or the internal properties of the required information system. This language is based on a block structure approach to provide data oriented control structures. Data selection by identifying subsets of data by their properties, creation of such subsets of data and data generation are developed as special features of this language.

Time has been used as a special concept. The control structure is used to identify different kinds of needs of updating so that the update aspects can be further utilised. The three different features of SDL, those of report description input document description and data transformation description, together can be used to completely describe the required information system.

In the proposed methodology, the SDL description is used to derive the data needs for the reports. An algorithm for doing this has been implemented. Further data needs are identified for carrying out various data transformations. The data availability from the input documents and from the data transformations is also derived. The needs are matched against the availability.

The method for doing this is described. This brings out the missing information and superfluous information.

In real life, systems need more detailed analysis. Ideas have been developed to analyse system descriptions in SDL, in greater detail to point out the shortcomings of the description. The structure of the language can also be used to collect information related to system design interactively from the user. How queries may be generated algorithmically to interact with the user for correcting the system description at a detailed level has been described.

In conclusion it can be said that the process of system analysis and description can be aided by the computer by using SDL which is aimed at describing data associations.

CHAPTER 1

INTRODUCTION

With the increased availability of computers, data has become a more available resource to control the activity of business and this has led to a tremendous increase in the development of management information systems of various kinds. Today, almost eighty percent of the total available computing power is being used for business systems and therefore they are of primary interest to the computer industry.

In near future potential users of data processing and new areas of industrial and administrative applications will generate an increasing demand for programming man-power. To keep such man-power requirements to a realisable level research is needed towards the automation of system design. Development of sophisticated software tools and advacement of design methodologies is envisaged to reduce the system design efforts and time. To cut down the system development time, enhancement in quality, speed, and ease of development is also necessary. The objective of this thesis is to develop such computer aids for system design of management information systems.

1-1. THE PROBLEM OF INFORMATION SYSTEM DESIGN

Systems like inventory accounting, personnel payroll, manufacturing control, accounting, air line reservations, banking etc.,

normally constitute the class of data processing systems. The process of system design at the current state-of-art level can be described in terms of five phases: problem acquisition, system analysis, system design, system generation or implementation and testing. At different stages of development, iterations over some of these phases become necessary due to ambiguity in specifications or due to erroneous analysis. A schematic diagram of this process, given in Figure 1-1 summarises the various feedback loops in this process.

In the problem acquisition phase the information needs of the organisation are identified. With the help of a dialogue between the user and the systems analyst, the information flow in the organisation is communicated to the systems analyst. A narrative documentation is created and the feasibility of a realistic design is studied.

In the systems analysis phase, with a basic design in mind, the data processing needs of the user's system are identified in greater detail and documented by using various kinds of forms specific to an installation. What can not be specified in forms is put down as a narrative. Quite often, reiteration of the problem acquisition phase is needed. These system specffications are considered as the beginning point for the design phase of this iteration.

System design phase is the evaluation of alternative designs, generation of program specifications, file specifications, and system control requirements in terms of flow charts etc. Most often the

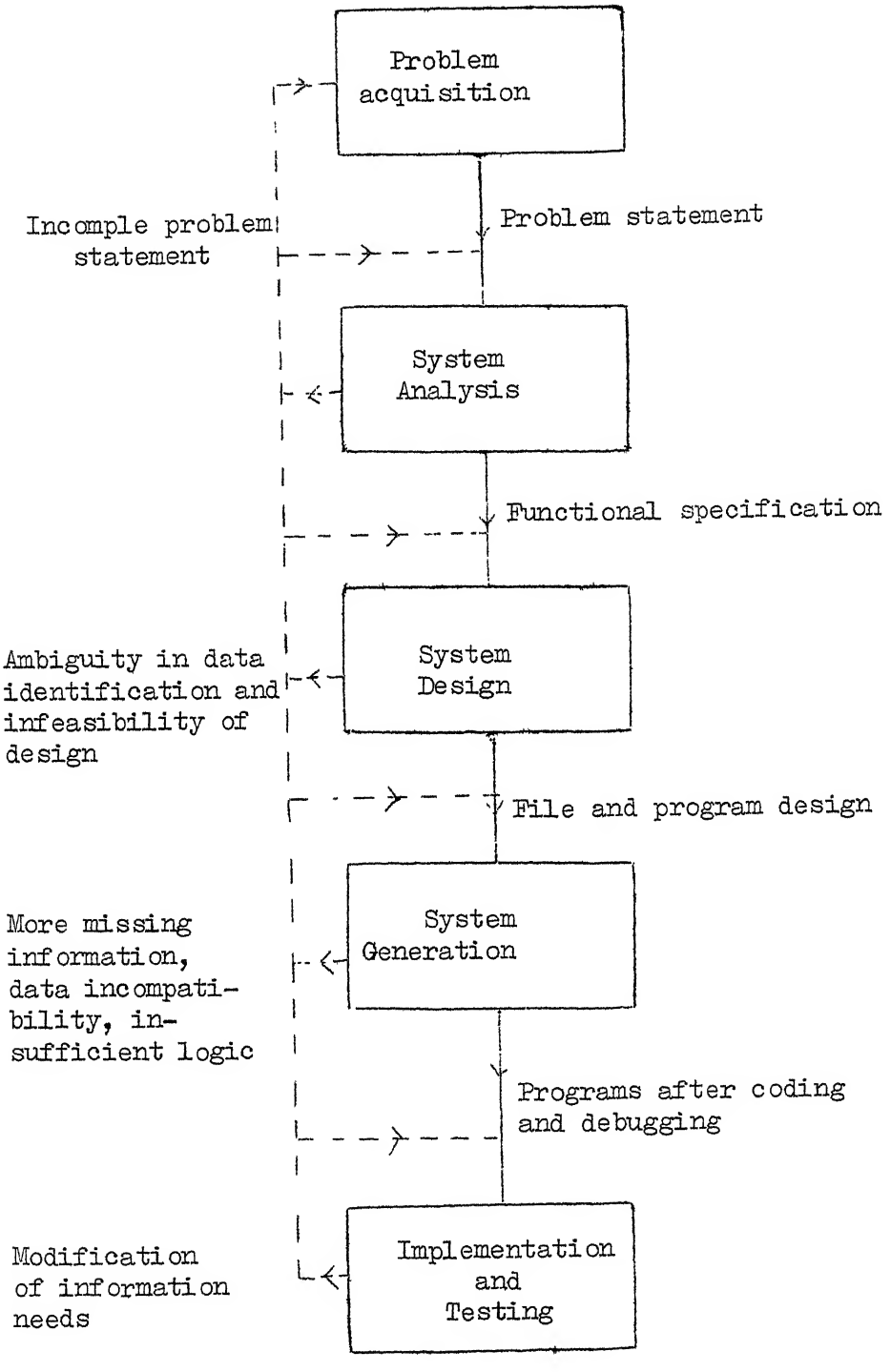


Figure 1-1. System Development Cycle

system designer is expected to be the same person as the systems analyst due to embedded ambiguity in the specification of the system.

Phases of implementation and testing require programming and operational work. These are normally the phases that require small iterations over the whole process of design due to various types of errors that are unavoidable during problem acquisition and analysis. An approximate break up of the technical work requirements [PRY-76] is as follows:

1. Problem statement	8%
2. Functional specification	22%
3. File structure and module design	15%
4. Program design, coding and debugging	25%
5. Integration and installation	30%

Attempts in current research are towards reducing the system development time and effort by introducing the computer in the information system design loop, in the form of an aid, and further, if possible, in the form of a basic designing unit. If the information needs of the organisation are specified in such a manner that the specifications have less errors and are more complete in a logical sense, then the number of iterations over the system development cycle will be reduced thereby reducing the time and effort required.

If the information needs can be recorded in a consistent and formal manner, then some of the work of analysis, design and generation can be automated. This would reduce the involvement and work required from the analyst or the programmer. Automated analysis can, in turn, interact with the user and help in creating a better and error-free statement of the problem.

1-2. IDEAS DEVELOPED IN THE THESIS

It has been observed that while recognising the need for a computerised information system the user is very clear about the outputs and reports expected from the system. A simple reason for this is the influence these reports and data outputs have on the decision making process besides the other benefits of a computerised system. At this stage the user may not have a clear idea of the process of obtaining these reports.

In this thesis a methodology is developed for identifying the information needs of the user, making reports as the starting point for the systematic analysis of the information needs.

A language for describing reports has been developed with special features for data generation and formatting related to the reporting medium. A block structure is followed in the language to allow structured development of the report description by the user. Algorithms are developed to identify data needs from the report descriptions. Input documents being a variation of reports, the user can also specify the data availability with the help of this language.

In business data processing, quite often, the data sets are partitioned into a large number of subgroups which undergo similar data transformations or processing. A language has been developed for describing such data transformations. Handling such subsetted data sets is a special feature of this language. Algorithms are developed to match the data needs for reports with the data availability from input and data transformations. The data descripancies, missing data items, and other analysis information reported by these algorithms can then be used to correct and modify the problem descriptions.

Algorithms have also been developed to further analyse the problem description for design purposes. Procedures for generating queries for analysing have been discussed.

1-3. OUTLINE OF THE THESIS

Chapter 2 of the thesis gives a survey of the research work currently being done in this area. In the first part the data oriented user-level languages and the form and power of these languages is described. Second part deals with the over all system design and some of the implemented systems. Difficulties encountered in various approaches have been summarised.

Chapter 3 of the thesis gives the basic approach of stating the information needs for a system by describing the report requirements. Different types of problems encountered in making formal report descriptions are discussed. A language is proposed for giving formal,

complete and user-oriented report descriptions. Syntax for such a language is developed. The semantics of the language is described with the aid of detailed examples. An implementation for getting data needs from report descriptions, is discussed.

Chapter 4 of the thesis describes the document and process description parts of the system description language. User can directly describe the data transformations and data availability in terms of documents, so that the data needs and data availability together can be used to identify the missing information. A concept of subsetted data sets has been developed as the basis of the system description. The syntax and semantics of the system description language is described. A generalised update concept and its relation to data cycles and data storage is discussed. This concept is incorporated in the language for system description.

Chapter 5 of the thesis describes the different interactive aids that can be used for system development if this methodology of system description is used. Aids to collect, design and analysis information by asking questions from the user are also discussed.

Chapter 6 of the thesis describes a detailed case-study to illustrate the need and the utility of various aspects of the system description language. This case study demonstrates the necessity for different kind of features in the language and the ease of use of the language.

Chapter 7 of the thesis summarises the conclusions and discusses further research work to be done in this area.

-

CHAPTER 2

AUTOMATION FOR INFORMATION SYSTEMS

Generalized business data processing was recognised as a research area in the early sixties [CDA 59] and work was started towards data system languages and theories of data processing. A number of concepts and aids for design and descriptions of such systems were developed over the decade [COU 74]. Around 1970, work on this problem was also started by various groups at Michigan, Sweden, Project MAC at MIT, IBM Research Laboratory and others.

2-1. DIRECTION OF RESEARCH

Research in this field is focussed on two different areas. Automation of system analysis and design is one of these areas where a number of systems have been implemented and are undergoing further development. ISDOS [TEI 71] , Hoskyn's system [RHO 72] , Protosystem I [RUT 75] are some of the examples which will be discussed in detail later in this chapter. Small commercial software like SAM [ADR 74] (system analysis machine), Meta COBOL [ADR 74], CONVERT etc., are available which incorporate the concepts of system analysis and data representation in a limited sense. However, these will not be discussed.

The second area of research is to develop the interface by which the user can describe his information requirements and information transformation; in other words the languages for user descriptions.

For the purpose of an overall study, the languages that are being currently developed may be grouped according to the basic approaches taken. Each group will be discussed in later section of this chapter. Query-by-Example, HI-IQ (HIerarchical Interactive Query), SEQUEL (Structured English QUEry Languages), FORAL and RIL (Representation Independent Language), the language for parameterisation of information systems etc., are based on the relational data base concept. BDL (Business Definition Language) is for questionnaire driven interaction for building application models. MAPL and OWL are high level languages for interactively building models for the world-of-business. Different types of naming mechanisms are used in these two languages to build up relationships. These two languages will be discussed along with the Proto System I.

ADS (Accuretely Defined Systems), PSL (Problem Statement Language), and TAG (Time Automated Grid), directly follow a data processing approach. These will be discussed along with the other aspects of ISDOS, because these have been implemented with ISDOS. Business system planning, OSD (Object System Descriptions), IA (Information Analysis), Systemeering (System Engineering), PIS (Product Information System) are some of the management level techniques, which are partially computerised. These will be discussed together, later.

2-2. DATA BASE ORIENTED LANGUAGES

A number of query languages which provide user interfaces for logical selection of data from the relational data bases also provide some features for making the queries more readable. The capability of these languages is similar to that of DSL-- [COD 72] described by Codd. The reasonsfor discussing these languages in the context of data processing are the following:

(1) Data processing systems are now designed with data base concepts, using data base software.

(2) These query languages are non-procedural in nature and user-oriented in their form. Data processing languages also seek these two properties.

(3) It is generally understood that a combination of fairly complex queries is equivalent to a data processing transformation. For example, SEQUEL as a data definition facility for an integrated data base[CHM 73] is required to encompass the functions of a general purpose query language to accomplish many of the tasks which were previously reserved for application programs.

Limitations of these languages will be discussed by taking DSL-/" as the standard. The form of these languages will be described first and then the capability of these languages will be discussed together.

2-2.1 FORM OF THE LANGUAGES

2-2.1.1 SEQUEL: It has a block structured form in which, by using WHERE SELECT and FROM clauses the need for range description is eliminated and by using a block structure the use of tuple variables for linking is avoided. This makes a statement more natural for a user. SEQUEL is implemented with a model using XRM relational memory system with the help of a PL/1 interpreter.

2-2.1.2 HIERARCHICAL LANGUAGE FOR INTERACTIVE QUERIES: HI-IQ used as an interface to Data Base Structure Designer [GER 76] is more powerful than SEQUEL due to its hierarchical structure. In this the selection is incorporated at every level. Statements in HI-IQ are less English-like but the aim of HI-IQ is to make statements to set the environment for data base structure design. HI-IQ is mentioned here, because of its power of selection while maintaining a fairly high level interface. This is also an implemented system.

2-2.1.3 QUERY-BY-EXAMPLE: QBE [ZLO 77] is the user interface for the System for Business Automation. The language is relationally complete. It uses a novel idea of using examples of concepts as synonyms of concepts in a local context. The concepts

here happen to be domain names, relation names, and instances of domain-values which are to be treated as separate entities. Also the external structure of the language is made up of forms. Therefore, instead of writing expressions and statements in a formal language the user is restricted to filling up forms. This provides ease-of-use and familiarity to user. Nesting of forms is also allowed. It is obvious that if a process or data to be described becomes complex, then description by structure is easier than description by instances or examples. Advanced form of this instance concept is expected to create problems similar to that of natural language processing.

2-2.1.4 RIL and FORAL: FORAL [SEN 75] and RIL (Representation Independent Language) [FEH 73] are higher level interfaces of DIAM II system. RIL is as powerful as the other relational languages and has more features for derivations, insertions, etc., which have been identified as essential for data manipulation. FORAL is the user interface for RIL aiming at allowing the user to represent binary and modified binary relationships between names-for-things. It is still under development.

2-2.1.5 A Language for Parameterisation of Information Systems: This language [WAN 73] gives a special consideration to repetitions, volumes and formats of outputs alongwith the other features common to such languages. For describing formats of reports, line-objects and their spatial placement are considered.

It recognises that specific instances of reports do not determine completely the semantics of reports. In a system for automatic generation of computer programs [PRY 75] similar difficulties about report descriptions have been observed. Further it has been said that in report formatting and specifications of messages, continuity and availability of physical space, new page symbols, etc., are to be identified. These are frequently data dependent. At this point it can be added that sometimes a few data items themselves are dependent on these conditions. New report specification methods for solving these problems are needed.

2-2.2 CAPABILITY OF THE LANGUAGES

To discuss the capability of these languages, in a general way, DSL-α [COD 72] will be considered a familiar representative of these. DSL-α again is a data base query language. The purpose of this discussion is to bring out the basic reason for DSL-α being found very limited for data processing descrptions. Let us take two relations. For example

Relation PARTS Domains PART #, SUB-PART

Relation ORDERS Domains SUB-PART, DATE

Let us also formulate a query: List all the PARTs where all the Sub-PARTs are ordered on the same DATE. This obviously needs data selection based on subsetted data set ORDERS subsetted for SUB-PARTs of the same part i.e., PART #. For doing this, PIPELINE concept with some host language statements will have to be used.

In some other queries one may need image-functions which are not supplied in the language. Also the need for features like PIPELINE and IMAGE functions is there because the language is limited to only two quantifiers $\forall$ and $\exists$. These quantifiers are the only ones available in the predicate calculus used as the basis for DSL- α. A need for being able to partition the data sets into subsets of tuples and to be able to describe functions of those subsets is recognised at this level. This will also be discussed in further chapters of this thesis.

2-3. APPLICATION SYSTEM MODELS-QUESTIONNAIRE APPROACH

An interactive business definition system [HAM 75] is being developed to describe user needs by interacting with application models in a question anwer mode. The language that is used for this approach is BDL, (Business Definition Language) which forms the user interface for ACS (Application Customiser Services) where experts can build models of applications by defining program fragments and associated questionnaire. BDL is to allow the user to introduce options which have not been offered by the model.

It can be said that this is an advanced and generalised approach to what are currently, application packages. Practicability of this method depends on how meaningful is representation, referencing, and piecing together of program fragments. Work related to model building in this approach is yet to take shape. The language BDL, which is not yet implemented, has five components as follows:

1. Document Description Component - DDC

2. Human Interaction Component - HIC

3. Device Linking Component - DLC

4. Document Flow Component - DFC

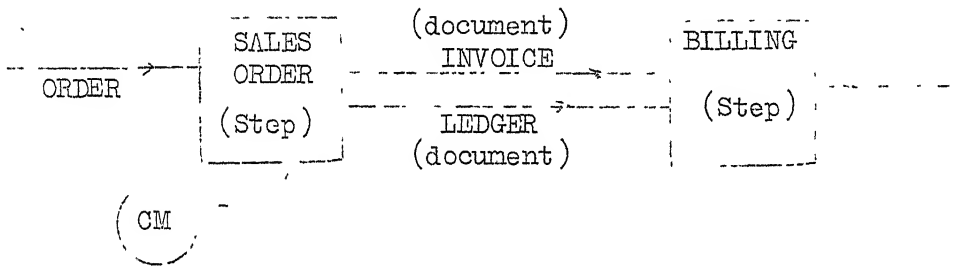
5. Document Transformation Component - DTC.

DDC is used to define the structure and attributes of documents. It is also meant for defining the format of the document in terms of media that will eventually be used for external realisation. It also allows definition of domain types like dollars, date, number etc. As DDC is based on a line-item concept (one line of the documents), it is suitable for forms and not for reports. Definition of external media and its effects are also not generalised.

HIC, the human iteraction component allows scope for user decisions in an interactively executing application. This is often needed for manually supervised application, for overriding supervisory actions. How HIC can be made a module and not a manifestation of the basic development of the application, can be understood only after the HIC and the model building process are fully developed.

DLC is for defining the linkage between the logical media described for documents and the actual combination of media used. Currently this function is being performed by terminal handling systems, written by the user, on generalised definition languages. DLC may be at a more logical level.

DFC is the document flow component for defining the basic structure of the application. It graphically represents the documents and the organisation units called steps, that use these documents.



It deals with steps, paths, documents and files. Files here mean sets of documents. Files are denoted by circles and access paths. A step executes when all its input documents are present. Accumulate, stream, copy, join (merge by arrival) are various types of step commands. It is mentioned that processing within steps, described later as DTC, is oriented around aggregates and therefore input to the steps should be groups of documents whenever possible (though groups may be singleton groups). It also expects the user to do explicitely all the linking. This suggests that there are difficulties in recognising and handling parallelism of steps. Real life document oriented systems have parallelism as a basic property. Then, it is more the form and not the capability of DFC, that is document oriented.

DTC as the document transformation component is the basic capability of BDL. DTC statements are like filling up a form under five headings of NAME, DOMAIN, GROUP, FILTER, DERIVATION and a heading-less space for conditions, whenever needed. The rows actually form part of a hierarchy shown by indentations which describes repeating groups in the output. Under the name column data items on the output are specified. The domain column specifies the name of the group (which is the input file name) which generates the data item. Putting TOTAL in the name column and one of the name column entries in the domain column is also allowed. How this can be linked up with the DDC is not very clear. Group column can have entries of data items which are used for lower levels. This allows very simple subsetting. Filter column is for accessing data-items from other files (practically a means of accessing master reference files), though only simple selection can be done on these files. Derivation column is for simple calculation and precoded (for example SUM, COUNT etc) set calculations.

In DTC, the power of the language has been compromised for a form-like appearance. Also, the form filling requires a lot of understanding, because of the power of language. At present it can not be said as to what extent it gives ease-of-use.

2-4. HOSKYN'S SYSTEM

This system was implemented in 1972 on an IBM system using HSL/1 (Hoskyn's System Language). The system generates COBOL programs

from user statements given in HSL/1. Besides a basic checking for undefined elements etc., this system does not perform any design activities. It plugs the various names for direct COBOL program generation.

HSL/1 is in the form of three tables, which are called matrices by HSL. These are program/file specifications, record/data specifications and condition/action specifications. Linking is done by allotting a column to each program by putting P in the row for that program. For files I/O/U etc., are put in the corresponding columns. For simple systems it practically does only the laborious work of writing data divisions. The same thing could be obtained by properly using the copy option of some COBOL compilers.

2-5. PROTO SYSTEM I

Proto System I [RUT 75] is being designed to automate the process of programming a problem, based on the understanding created by an English language discourse, specifically for management information systems. In this system, the process of writing programs is modelled as a sequence of translators from an English language task specification to machine code. The different levels are described as follows:

T0: Problem acquisition (English → OWL)

T1: Global problem solution (OWL → DSSL)

T2: Algorithmic solution (DSSL → CDSL)

T3: Computational (CDSL → PL/I and JCL)

T4: Compilation and loading (PL/1 and JCL → machine executable)

Specifications of the problem, in terms of data processing functions is available at DSSL level and most of the system design and optimization of this kind is performed at T2 level. Translations of the kind TO and T1 are suitable for custom made applications where domain dependent terms are taught to the translator (for example level T1) by an expert who can represent the informational view of the problem while presenting it in domain dependent terms. In a very general sense, this is also what a systems analyst does. In this case the computer aid available to the analyst or the problem expert is a system, where world-of-business models can be built.

Information on OWL is not available but a related language MAPL [MAR 73] can be used to discuss the level of model building. The basic constructs are relationships. For example a-kind-of is a relationship which describes objects as characteristics of objects: ORDER a-kind-of DOCUMENT. Time, activity, data-item, occurrence etc. may be some of the basic constructs needed for business models. To give an idea, for a very small real life sales system, one may have to construct some 4000 concepts before a user's problem in that domain may be stated. Levels higher than DSSL may be suitable only for package applications.

Major concepts of DSSL (Detailed System Simulations Languages) leve are timing (period), aggregate data objects, computations and initial values. For computations of subsets, DSSL also gives a few built-in functions like SUBSETMAX, SUBSETPLUS, etc. A variable is specified

as a combination of variable name, time period and keys. Though a variable version of current time cycle of given period and that of the previous period is allowed, no other versions are allowed. Naturally, this is a requirement for update computations. Precedence of computations in DSSL is implicit in data relationships and therefore user generated subset functions cannot be represented. This is a limitation for general data processing. Further development of the system, however, may incorporate such facilities.

At the design and generation level the system has a simulator, a translator, and an optimiser. Translator translates from the compact form of DSSL to more algorithmic forms while maintaining all the design options. It also computes properties like predicates, inputs, outputs, precedence, keys of data items as well as computations. Optimiser does the aggregation of computations as programs and that of data items as data sets. To check the multiplicity and existence of data items for estimating I/O costs, the optimiser interacts with the simulator. Simulator directly executes that part of DSSL description (about which an estimate has to be made) on initialised data items. It collects information about maximum values and the probabilities that certain predicates will be true. Aggregate system behaviour is obtained by executing parts of the system if one data item of each type is given to begin the system.

At this point it can be said that part of the aggregate informations, obtained from the simulator, is at times well known to the application user. Interactive aids may be developed to make the user give such information because such simulation may become a very costly and cumbersome process when developed to its usable level. It may be mentioned that optimiser avoids problems of combinatorial order, by combining only two processes or two data aggregates at a time.

2-6. ISDOS : INFORMATION SYSTEM DESIGN AND OPTIMISATION SYSTEM

A system for automatic design of information systems [TEI 71] has been developed and implemented on UNIVAC 1108, IBM 360 and CDC 6500 computers. At the user interface it uses TAG [MYE 63] (Time Automated Grid) or ADS [NUN 74] (Accurately Defined Systems). At the analysis level it has the problem statement analyser, PSA [HER 74] with the problem statement language PSL. At the design level it has a system optimisation and design algorithm SODA [NUN 71] which has two modules ALT and OPT for design alternatives and optimisation respectively. The SODA/ALT generates alternatives by using program and file structure algorithms. SODA/OPT has algorithms for optimisation and performance evaluation.

TAG is a system design aid which analyses the data item relationships in the frame of time, by accepting information on an input output analysis form. All the input, output, and history data sets used in the system are listed along with information about the volume, frequency, and time of use and the various data items that constitute

the set. Data sets can be given priorities within a particular time cycle. TAG gives an analysis of the use of data items in the increasing time cycle order. This technique was originated to consolidate the information for manual analysis.

The ADS technique uses comparatively more structured set of five forms. Specifications for inputs, outputs, history, computations and conditions are listed on separate forms and the linking of data items is established by specifying the line numbers and form or page numbers. These forms were also designed for manual system documentation. As the capability of ADS is limited to element by element processing generally used in sequential systems there is no way of specifying the structure of the problem and data.

SODA/PSL, the language acceptable to the analyser module of SODA/PSA, is a very high level language for specifying units or modules of the system. It has features for naming the parameters of the unit. For example, units could be inputs, outputs, conditions, process at a lower level, process at a higher level, frequencies, and volumes etc. It is very suitable for well developed pieces of software or problem statement units being handled by different people. It would need too many explicit statements about the properties of modules. for the first level descriptions of systems.

Analysis of the total problem is performed by SODA/PSA by using matrix techniques [LAN 65] for analysing data definitions, timings, precedences, and volume etc. It also performs more detailed analysis

to generate suitable forms of information for design algorithms. It gives the error diagnostics but does not aid the process of analysis by directing the user towards expected error areas or towards giving more information for design purposes.

At the design level, the alternatives for design of programs and files are generated by SODA/ALT and evaluated by using transport volume as a performance criteria and using branch and bound methods for different CPU's, core sizes, auxiliary memory and file organisations. This approach leads to combinatorial problems. A more heuristic approach may be more usable.

2-7. THEORETICAL ANALYSIS OF INFORMATION - for business organizations

Some approaches to the theory of information systems [LAN 63] are based on the analysis of current functions of the organisation. Here, it is assumed that 'functions' of the firm or the organization, must be in action, for it to fulfill its objectives. Defining these functions is one way of stating these objectives. Generic information system design approach [CHA 74] considers information needs by identifying and analysing all the documents and the data-items that are active in the organisation. Systemeering is [LUN 75] another direction in which work is being done to analyse the information needs of administrative systems, by describing the system in terms of material flow and message flow. The language in this case uses a graphical method. The procedures for doing information component analysis and information precedence analysis etc. are being worked out.

At present it can be said to be a formal method of describing the basics of the business functions. The problem of isolating the information flow for decision making and organisational control and then representing it in an accurate way has yet to be worked out in these approaches.

2-8. CONCLUSIONS

Characteristics and developments of different systems which have been discussed in the previous sections of this chapter are summarized in Table 2.1 and Table 2.2. Description, analysis, and design of information systems for business data processing are very much inter-related. It has been seen that development of one stage in isolation, limits the capability of other stages. It is felt that an integrated approach will prove more successful. A few observations made at the user interface level may be reiterated as follows:

(1) Ease-of-use may be enhanced by making the environment interactive, and the language more like English. However, the latter may pose semantic difficulties and therefore atleast some commonly used associations related to data may be introduced.

(2) Description languages should be more data oriented in the sense of dealing with sets and subsets of data.

(3) Reports, messages need more complete and detailed descriptions as they are the major vehicle of communication between the user and the information.

Systems analysis is more mechanical than itelligent at present. It should be recognised that the main shortcomings of the problem descriptions creep in when the user tries to make a data oriented statement while his mental model is still materialistic. For example, one does not remember while describing sales, to update the store for commodities which have been sold out though this may not happen while describing stores inventory or accounting. The analysis should also guide the user in this sense.

Techniques of matrix algebra and simulation etc., are currently being used for system design. Heuristic approaches may be more workable. When the user describes different business functions, the organisations and the sequence of descriptions tends to be similar to the division and arrangement of working in the organisation. Scope of changes in the information system in future due to ommission or changes in the needs will be very much related to this inherent structure and systems can be made more adaptable if the inherent problem structure can be put to use in design.

-

TABLE 2-1. Systems for Design and Description of Information Systems

System	Proto System I	ISDOS	Hoskyn's System	Application Questionnaire	Data Base Oriented Systems
Languages	OWL, MAPL	ADS, TAG PSL	HSL/1	BDL	QBE, FORAL RIL, SEQUEL HI-IQ
Type of Language	English-like	Data Processing	Matrix	Form Processing	DSL-
System Description	Model building using binary relationships	Input, Output, process relationships of small units	Associations of programs files and data by matrices	Model building by application question naive	Queries and data manipulation
Method of analysis	Cross checking	Cross checking	Cross checking	Not known	Not relevant
Selection of Design Alternatives	Simulation-based	Combinatorial generation	User's problem statement	Not known	Not relevant
Implementation	Yes	Yes	Yes	Not known	Yes

(continued)

System	Proto System I	ISDOS	Hoskyn's System	Application Questionnaire	Data Base Oriented
Being Developed further	Yes	Not known	No	Not known	Not relevant
Basic aim of the system	Generalised inventory model	Handling and generation of large software	Generation of small data processing systems	Generalised application package building	Data definition and manipulation for queries
Existing problems	User's model building	Time complexity of design	Limited capability	Assembling the model together	Limited query/problem formulation

Table 2-1 (continued)

TABLE 2-2. Data Base Oriented Languages

Language	Query-by-Example	SEQUEL	FORAL/RIL	HI-IQ
Implementation	System for Business Automation	System R	DIAM II RIL only	Data structuriser
Form of the language	Form, filled in with examples	Block structured query	English like statements (in FORAL), queries (RIL)	Hierarchical query statements

CHAPTER 3

DATA NEEDS FOR INFORMATION SYSTEMS

Use of an integrated information system introduces a change in the information flow within the organisation. Different types of information may be available from such an information system, such as, usage reports which help in the execution of functions; statistics or status reports for checking and control; study reports in response to queries etc. How much of such information can be useful for the organisation and what it implies in terms of diverting information to the machine and how much it costs has to be visualised. Therefore the role of the information system has to be well understood before the information needs can be described.

3-1. IDENTIFICATION OF INFORMATION NEEDS

Information for administrative control is used in the form of documents and reports, whereas, other systems like real time systems and control systems, channelise information, more often by message display, transducer and converters etc. The methodology developed in this thesis is based on the fact that people use information in the form of documents and reports of various types. The primary aim of an information system is to provide such reports. If the requirements of these reports are very well known, by way of their content, and form, then the user's problem, i.e., the

requirements of the information system, can be well stated. This, in turn, may be utilised to find out the processing and data required to solve the problem.

In this way reports also limit the user's expectations from the information systems to whatever may be obtained in the form of reports. This forces the user to set the degree of ambition of the design of the system before going into the details of processing and design. Gross requirements about the environment of processing, types of outputs devices needed, etc., also emerge at this stage. Therefore the role of the information system becomes well identified at the beginning of the system development cycle.

A formal description of the problem of information system design is needed before the computer can be used for solving it. Different methods of such descriptions like model building etc. have been discussed in Chapter 2 alongwith the problems associated with those methods. A simpler, but otherwise powerful approach, is being developed in this thesis where a description of the outputs or the reports, will be treated as the basic problem statement. A block diagram shown in Figure 3-1 describes the approach.

Reports can be fully described in a language which will be discussed in the next section. Such a formal description is analysed to find the data needs for all the reports. User can describe the input information and the required information processing, i.e., the transformations, in a data oriented language which will be

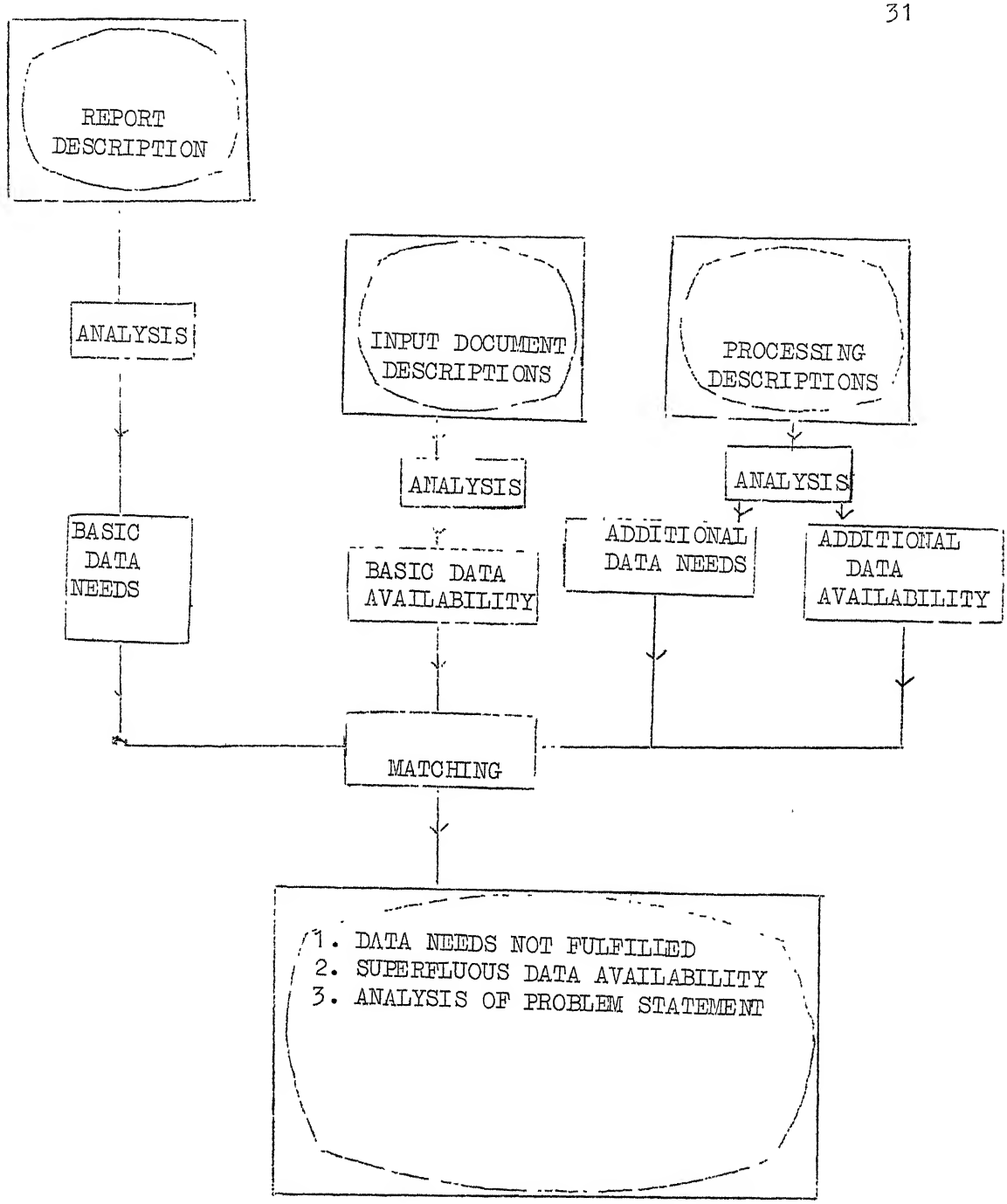


Figure 3-1. Information System Description

discussed in the next chapter. This is compared with the data needs, and any discrepencies or missing information is brought to user's attention. This process is repeated till a consistent and complete description of the information system is obtained.

3-2. REPORT DESCRIPTION LANGUAGE

Data in reports is formally recorded data in the context of information system design and is different in nature from informally recorded reports. Type of such data may be different such as numbers, words, pictures, lines, colours, signs, crosses, tick marks, seals and stamps, punches etc. Special requirements of non-alphanumeric data are not being considered while discussing the structure of reports.

3-2.1 STRUCTURE OF REPORTS

Reports have much more structure than messages, forms, registers, and other paper documents. Display of information on reports or documents has three important aspects:

(1) The actual instance of the data, its value and size

(2) Name of the data item which can uniquely identify the data for the object system.

(3) Heading or a name for relatively local context.

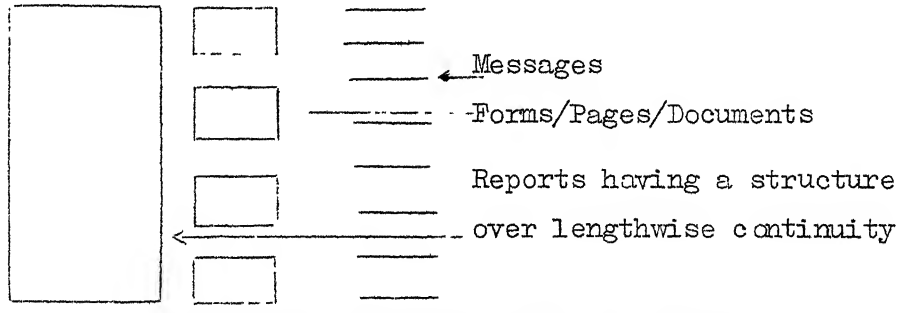


Figure 3-2. Different Type of outputs.

In the form oriented report languages or descriptions, such as, BDL or ADS, the dichotomy of data and its format is such that it tends to restrict at least one of the three, i.e., the data instance, data name or data heading. For example, in report description forms of Accurately Defined Systems, the relevant data items have to be associated with the format by linking up of line numbers in the forms. In report program generators, control breaks for certain data items is the only kind of structure that can be incorporated. In Business Definition Language, simple hierarchical structure of data items is used. Therefore a language is needed which has a comparatively free format for structuring data definition and format together in a way suitable for business systems.

Business systems have data which records properties of large sets of similar items. A unique name for selected data is not explicitely available and therefore it is made up by attaching qualifying names which belong to more aggregate levels. For example if there are 20 values for ORDERED-QUANTITY then one value may be expressed by ORDERED-QUANTITY of a COMMODITY for a CUSTOMER. As the heading has limited expressibility, structure is given to the report so that a qualifier, common to a number of instances, may be needed only once. The user is, often, very specific about the format.

For these reasons it can be said that a language which supports hierarchical naming of data will be more suitable for report descriptions. The language described in Section 4 of this chapter is based

on a block structure concept which can maintain hierarchy of levels. This hierarchy is also utilised for effective data naming. Since the semantics of the report may not be completely determined by specific instances of the reports, it can be described along with the structure. Here the semantics of the report means conditional appearance and formatting of data-items. For example, if the 'description' is to be printed only when first time the 'code' appears on our output, or when for selected values of data items special action is to be taken.

Example 3-1 illustrates a situation where report is selection based. Though the syntax of the description language will be introduced in later sections of this chapter, the example makes use of the language.

3-2.2 STRUCTURE AND THE MEDIUM OF REPORT

To isolate regions of common qualifiers of data and to get speed of referencing for those who are to use the report; continuity and availability of physical space, line and column spacings, indentations, new pages etc., are to be identified. The syntactic structure of the formatted report is captured by the spatial placement of lines of data. Both these are data dependent. Therefore some method of synchronising the conditions on the medium of reporting, or the output device and the data being put on it, should be available.

Example 3-1.

REPORT ON INVOICE EDITING PAGE 1

PRODUCT CODE	INVOICE NO	CALCULATED VALUE	PRODUCT VALUE	MAXIMUM DISCOUNT	DISCOUNT VALUE	COMMENT
0513	10	53.5	53.0			COMMENT-1
0600	1			10	12	COMMENT-2
0713	3	60.0	65.0	9	15	COMMENT-1
						COMMENT-2
0613	6	23.5	23.0			COMMENT-1
....	..	....	....	..	..	.....
....	..	....	....			

```
FOR EACH DAY REPORT INVOICE-EDIT
FROM DATA SET INVOICE-AUDIT:S
BLANK 12, 'REPOIT ON INVOICE EDITING," PAGE-NUMBER
NEW-LINE, HEADING-LINE-1, NEW-LINE, HEADING-LINE-2
FOR EACH OF THE INVOICE-AUDIT BEGIN
  FOR DISCOUNT-ABOVE-MAX = 1 OR VALUE-DISCREPANCY-COUNT = 1 BEGIN
    NEW-LINE, PRODUCT-CODE, INVOICE-NO
    FOR VALUE-DISCREPANCY-COUNT = 1 BEGIN
      CALCULATED-VALUE, PRODUCT-VALUE END
    FOR DISCOUNT-ABOVE-MAX = 1 BEGIN
      MAXIMUM-DISCOUNT, DISCOUNT-VALUE END
    COMMENT-1
    FOR COMMENT-1 = BLANK BEGIN COMMENT-2, END
    FOR COMMENT-1 ≠ BLANK AND COMMENT-2 ≠ BLANK BEGIN
      NEW LINE, COMMENT-2 END
  FOR NEW-PAGE BEGIN PAGE-NUMBER END
FORMAT HEADING-LINE-1 FROM 1 VALUE "PRODU"-
-"CT INVOICE CALCULATED PRODUCT MAXIMUM DISCOUNT COMMENT"
HEADING-2 FROM 2 VALUE "CODE        NO."-
-"      VALUE   VALUE DISCOUNT VALUE"
PRODUCT-CODE FROM 2  SIZE 4  INVOICE-NO FROM 12  SIZE 2
CALCULATED-VALUE FROM 24 PICTURE 99.9  PRODUCT-VALUE FROM 36 PICTURE 99.9
MAXIMUM-DISCOUNT FROM 45 SIZE 2  DISCOUNT-VALUE FROM 55 SIZE 2
COMMENT FROM 60 SIZE 10
OVER
```

Special features have been developed in the report description language to sense the conditions and to control the reporting device in a logical sense. This is done by taking paper as the output medium in general and incorporating a two dimensional forward control on it and associating a current status with it.

A simple example of this is when a variable number of lines appear on an invoice with a three line 'end of invoice' procedure; suppose the end-of-invoice part is to be printed at the end of the page unless the last page of the invoice contains only end-of-invoice data the language is shown in Example 3-2.

Example 3-2.

```
REPORT INVOICES FROM DATA SET BILL:S
FOR EACH CUSTOMER OF BILL:S BEGIN
    NEW-PAGE, CUSTOMER
    FOR EACH OF THE BILL:S BEGIN
            NEW-LINE, BILL-NO, AMOUNT, END
    FOR LINE-NUMBER ≠ 1 BEGIN
          3 LINES BEFORE END-OF-PAGE END
    "END OF INVOICE"
    NEW LINE, "SIGNATURE"
    NEW LINE, "........."
OVER
```

3-2.3 DATA AND THE REPORTING MEDIUM

From the report point of view, there are two types of data items present on the report.

1. Those which are directly obtained from the data base
2. Those which are dynamically dependent on the synchronisation of format with other data items.

A simple example is that of carry-forward-totals used in accounting applications. Such items are included in the reports to give speed and ease of referencing to the people. To give a different example, in listing out customer-wise orders, only for the first order, the customer name need be printed unless it is a new page. Such requirements are there to reduce the redundancy of information.

If the format description of reports and the processing of the report-information are described in a segregated manner then such situations are difficult to handle. In this approach, status conditions of the reporting medium and generation or selection of data items can be done together.

Example 3.3

```
REPORT EMPLOYEE-DETAIL FROM DATA SET EMPLOYEE
FOR EACH DEPT-NO BEGIN
    NEW-PAGE, DEPT-HEADING, DEPT-NO
    NEW-LINE, "DEPT-TYPE" "DEP-CLASS"
    NEW-PAGE, MAKE CARRY-FORWARD-BALANCE = 0
    FOR EACH NEW-PAGE BEGIN
        PAGE-HEADING,PAGE-NUMBER, CARRY-FORWARD-BAL END
    FOR EACH EMPLOYEE-NO BEGIN
        NAME,CLASS,ADDRESS
        NEW-LINE, BASIC, NET
        ADD NET TO CARRY-FORWARD-BALANCE END
    FOR EACH 3 LINES BEFORE END-OF-PAGE BEGIN
        "TOTAL-BALANCE", CARRY-FORWARD-BALANCE END
    FORMAT PAGE-HEADING VALUE "EMPLOYEE-DETAIL" CARRY-FORWARD-BAL
     SIZE 8,DEPT-HEADING VALUE "THE DEPARTMENT" NAME SIZE 10 CLASS
    SIZE  10 ADDRESS SIZE 10 BASIC PIC 9(4).99
     NET PIC 9(6).99
     OVER
```

3-3. DESCRIPTION OF THE LANGUAGE

The report description in the proposed language starts by naming the report and naming a data stream from which the data is to be selectively put on the report. The user at this point need not visualise the exact data content of the data stream. It is more for the sake of establishing a local reference to a data set which could eventually be used for report generation. Also, it need not be a defined data set, as the reports are the starting point of the description. The logical structure and the physical format of the report are then described in an interleaved manner. The language is a free format language.

3-3.1 FEATURES FOR STRUCTURE DESCRIPTION

The report structure may be developed in a step by step fashion. A report may be described in terms of subreports and group names may be used for data items which are functionally similar. For example, EARNINGS : BASIC, D.A., HRA etc. Such groups and subgroups may be described at a later stage. This is illustrated in the Example 3-4.

Detailed description is given in the form of blocks and nesting of blocks at different logical levels. This facilitates special descriptions at the beginning and at the end of a group of data. In present report generators a logical level may be created by a change in the value of a data item. With this kind of block structure different type of conditions may be used to create a level. Depending on the condition which created a level different description for the report at that level may be given.

Example 3-4.

```
REPORT COINAGE-PAGE FROM DATA SET COIN-RECORD
NEW-PAGE, PAGE-NUMBER, HEADING-1, COIN-1, COIN-2, COIN-3, COIN-4
NEW-LINE, COIN-VALUE-HEADING, COIN-VALUE OCCURS 7 TIMES
TOTAL OF COIN-VALUE,
NEW-LINE, ALL "_ _"
OVER

REPORT DEPT-COIN-RECORD
FROM DATA SET DEPT-COIN-RECORD
FOR EACH DEPT-COIN-RECORD BEGIN
    COINAGE-PAGE REPORT USING (DEPT-COIN-RECORD FOR COIN-RECORD) END
COINAGE-PAGE REPORT USING (TOTAL OF DEPT-COIN-RECORD FOR COIN-RECORD)
OVER

FOR EACH MONTH
REPORT COINAGE
FROM DATA-SET EMPLOYEE-COINAGE

FOR EACH DEPT BEGIN
    FOR DEPT   32 BEGIN MAKE PAYPOINT OF EMPLOYEE-COINAGE = 1 END
    FOR DEPT   40 BEGIN MAKE PAYPOINT OF EMPLOYEE-COINAGE = 2 END
    FOR DEPT   61 BEGIN MAKE PAYPOINT OF EMPLOYEE-COINAGE = 3 END
    FOR DEPT   99 BEGIN MAKE PAYPOINT OF EMPLOYEE-COINAGE = 4 END
    END
FOR EACH PAYPOINT BEGIN
    FOR EACH DEPT BEGIN
        DEPT-COIN-RECORD REPORT USING (EMPLOYEE-COINAGE FOR DEPT-COIN-RECORD
        END
    COINAGE-PAGE REPORT USING (TOTAL OF EMPLOYEE-COINAGE FOR COIN-RECORD)
    END
COINAGE-PAGE REPORT USING (TOTAL OF EMPLOYEE-COINAGE FOR COIN RECORD)
NEW LINE
ALL "_ _"
OVER
```

3-3.2 FEATURES FOR DATA SELECTION

Format of the report may be dependent on the properties of data. In ADS and report generators this is incorporated to the extent of control breaks and totals. In our language it is possible to specify properties of data in greater detail. For example if data item 1 is

non-zero only then certain other data items should be put on the report. A more important aspect of data selection, which is sub-setting of data, will be described in detail in the next chapter.

Data selection becomes more important when certain kind of processing for data generation has to be described specially in the context of reports due to its relation with the reporting medium.

3-3.3 FEATURES RELATED TO REPORTING MEDIUM

Status conditions of the reporting medium or the hardware, like NEW-PAGE, END-OF-PAGE etc., can be used alongwith the data conditions to format the report. Control of the reporting medium is also needed for formatting for selective positioning of data on paper and this may be described along with the data items and other processes. It helps in making the report description complete and independent.

3-4. SYNTAX OF THE LANGUAGE

Syntax of the elementary form of the language is being given here to aid the description given in the above sections.

```
<REPORT> ::= REPORT <REPORT-ID> <DETAIL> OVER
<DETAIL> ::= <STRUCTURE> <FORMAT> | <COPY-STATEMENT>
<STRUCTURE> ::=                    <DATA-SET-CLAUSE><PROCEDURE-BLOCK>

<REPORT-ID> ::= <VARIABLE-NAME>
<DATA-SET-CLAUSE> ::= FROM DATA SET <DATA-SET-NAME>
<DATA-SET-NAME> ::= <VARIABLE-NAME> | NIL | <VARIABLE-NAME><TIME-INSTANT>
<PROCEDURE-BLOCK> ::= <SIMPLE-PROCEDURE> |
                      FOR <CONDITION> BEGIN <PROCEDURE-BLOCK> END |
                      <PROCEDURE-BLOCK><PROCEDURE-BLOCK>
<SIMPLE-PROCEDURE> ::= <REPORT-HARDWARE-PROCEDURE>
                       <DATA-ITEM-STRING>
                       <CALCULATION-STATEMENT>
                       <REPORT-CALL>
<CONDITION> ::= <REPORT-HARDWARE-CONDITIONS>
                <DATA-SET-ELEMENT-SELECTION>
                <SIMPLE-DATA-CONDITIONS>
```

```
< REPORT-CALL > ::=< VARIABLE-NAME > REPORT |< COPY ST TEXT >
< USING-CLAUSE > ::= <EMPTY > | USING   REPLACE-CLAUSE
< REPLACE-CLAUSE > ::= < REPLACE-CLAUSE>  < REPLACE-CLAUSE > |
                       < ALPHA STRING>  FOR < ALPHA STRING >
< DATA-ITEM-STRING>  ::=  <DATA-ITEM > | <DATA-ITEM>  < DATA-ITEM-STRING>
< DATA ITEM>  ::= <SIMPLE-DATA-ITEM> | <CALCULATED-DATA-ITEM >
< CALCULATED-DATA-ITEM> ::=  TOTAL OF < VARIABLE-NAME >  |
                             COUNT < VARIABLE-NAME>  AS < VARIABLE-NAME> |
                             MAX< VARIABLE-NAME> AS < VARIABLE-NAME>  |
                             (Other standard functions may be added)
< SIMPLE-DATA-ITEM>  ::= <SINGLE-DATA-ITEM> | <ARRAY-  -DATA-ITEM >
< SINGLE-DATA-ITEM>  ::= <VARIABLE-NAME > |<LITERAL> | <OTHER-DI >
< ARRAY-DATA-ITEM>  ::= < VARIABLE-NAME > OCCURS < INTEGER > TIMES
< CALCULATION-STATEMENT>  ::=  MAKE <VARIABLE-NAME >  = < SINGLE-DATA-ITEM>
                               (assignment)
                               ADD< ADD-CLAUSE> |
                               COMPUTE < COMPUTE-CLAUSE > |
                               INCREMENT  < INC-CLAUSE> |
                               MULTIPLY  <MUL-CLAUSE>
 <REPORT-HARDWARE-PROCEDURE>  ::=  NEW-PAGE | NEW-LINE | PAGE |
                                   SKIP-TO-CHANNEL <INTEGER> |
                                   SKIP-TO-LINE< INTEGER> |
                                   SKIP  < INTEGER >LINES  |
                                   AFTER  < INTEGER > LINES | < INTEGER>
                                   BEFORE  END-OF-PAGE  |
                                   SET-PAGE <INTEGER>  LINES |
                                   SPECIAL-FORM-STATIONARY | REPRINT-LINE  |
                                   END-OF-PAGE
                                  <INTEGER>  COLUMNS OF SIZE< INTEGER>
                                   FROM <INTEGER>  |    ...
< REPORT-HARDWARE-CONDITIONS > ::= PAGE | NEW-PAGE| LINE< INTEGER >  |
                                   CHANNEL  < INTEGER > | END-OF-PAGE  |
                                  < INTEGER>  LINES BEFORE END-OF-PAGE  |
                                   PAGE-COUNT  < INTEGER > |
                                  <  INTEGER>  TO <INTEGER>  LINES BEFORE
                                  <  END-OF-PAGES>  | ...
< DATA-SET-ELEMENT-SELECTION>  ::= < GIVEN-VALUE-CONDITION > | < RANGE-CONDITION >
                                  | < UNIQUE-VALUE-CONDITION > |
                                    <UNIQUE-VALUE-CONDITION>  <SEQUENCE > |
                                    <ELEMENT-CONDITION >
< GIVEN-VALUE-CONDITION>  ::=  EACH  < VARIABLE-NAME > = < SINGLE-DATA-ITEM>
< UNIQUE-VALUE-CONDITION>  ::=  EACH< VARIABLE-NAME >
< ELEMENT-CONDITION > ::= EACH OF < DATA-SET-NAME > | < INGEGER > TIMES
< RANGE-CONDITION>  ::= EACH  < VARIABLE-NAME>  = ( < RANGE > )
< RANGE > ::=  < RANGE > , <RANGE> | = <SINGLE-DATA-ITEM >
               = FROM <SINGLE-DATA-ITEM > TO < SINGLE-DATA-ITEM >
< SEQUENCE  > ::=  ASCENDING|  DESCENDING| SEQUENCE ON < VARIABLE-NAME><SEQ-
< SIMPLE-DATA-CONDITION>  ::=< SIMPLE-CONDITION  > |                    UENCE>
                             < SIMPLE-DATA-CONDITION>< LOGICAL-RELATION>
                            -< SIMPLE-CONDITION>
< OTHER-DI >  ::= PAGE-NUMBER  | PAGE-COUNT  | LINE-NUMBER | <FORMATTED-DI>
<FORMATTED-DI>  ::= <VARIABLE-NAME> (PICTURE)  |  SPACE ( INTEGER )
```

<LOGICAL-RELATION> ::= AND | OR | NOT
<SIMPLE-CONDITION> ::= <VARIABLE-NAME><RELATION><SINGLE-DATA-ITEM>
<RELATION> ::= > | < | = | NOT = | ≯ | ≮ | ≥ | ≤
<FORMAT> ::= FORMAT <FORMAT-DETAIL>
<FORMAT-DETAIL> ::= <LINE> | <LINE> <FORMAT-DETAIL>
<LINE> ::= <DATA-REFERENCE> | <DATA-FORMAT> | DATA-POSITION
<DATA-FORMAT> ::= <PICTURE-CLAUSE> | <VALUE-CLAUSE>
<DATA-REFERENCE> ::= <VARIABLE-NAME> |
<VARIABLE-NAME> (<POSITION-IN-REPORT>)
<POSITION-IN-REPORT> ::= <INTEGER>
<PICTURE-CLAUSE> ::= AS IN COBOL | SIZE <INTEGER>
<VALUE-CLAUSE> ::= VALUE <LITERAL> | VALUE <TIME-INSTANT>
<POSITION-CLAUSE> ::= FROM CHARACTER <INTEGER> FROM <INTEGER> |
FROM COLUMN <INTEGER>
<VARIABLE-NAME> ::= <LETTER><REST-OF-THE-NAME>
<REST-OF-THE-NAME> ::= <LETTER> | <DIGIT> | <REST-OF-THE-NAME><REST-OF-THE NAME>
<INTEGER> ::= <DIGIT><DIGIT><INTEGER>
<DIGIT> ::= 1/2/ ... / 9/ 0
<LITERAL> ::= " <ALPHA-STRING> " | ALL " <ALPHA-STRING> " NON-BLANK BLANK | ZERO | SPACE | SPACES | ZEROS | <LITERAL> | <LITERAL>
<ALPHABET> ::= | A | B | C | ... , ... | <DIGIT>
<ALPHA-STRING> ::= <ALPHABET> | <ALPHABET> <ALPHA-STRING>
<LETTER> ::= A | B | C | ... | Z

3-5. IMPLEMENTATION FOR DATA NEEDS FROM REPORTS

From a report description given in this language, data requirements can be derived. A subset of the above syntax has been implemented, using a software package to generate the processor. The syntax is described in a prescribed way. The semantics is described in the form of COBOL procedures to be executed for the corresponding grammatical entities. The software package generates a COBOL program, incorporating the given semantics and syntax, complete with a generated scanner etc. Once the report description is found to have correct syntax the data needs are derived by going through the hierarchical relationships between various data-items. Data-items having constant values or values ·

generated in the course of the development of the report are dropped from the analysis. Data needs are described in the form of relations giving keys and attributes. Various data-items having the same keys for a unique identification are put as attributes in the same relation. Data-items occuring as keys for some attributes may themselves be attributes in other relations.

Here, attributes are those data-items which are to appear on the report. Keys are those data-items which are used to select and identify relevant values of attributes and data-items, at various levels, from a large set of data. Data needs for all the reports can then be analysed together. The part of the program which was input to the package is given in Appendix 1

When a suitable system software and configuration is available this implementation can be easily modified to change the input to interactive mode. In that case the errors can be pointed out and corrected on the spot. The output can be presented to the user in a more readable way by naming relations using keys. For example

Relation : 01;keys : CUSTOMER;attributes : NAME, ADDRESS

Relation : 02;keys : COMMODITY, COST;attributes: SIZE, COLOUR, BRAND

Relation display:

CUSTOMER-DETAIL (NAME, ADDRESS)

COMMODITY-COST-DETAIL (SIZE, COLOUR, BRAND)

3-6. CONCLUSION

In conclusion it can be said that identifying user's information needs is the first step towards information system design and analysis. It is very important to set the scope and the necessity of the data requirements. By identifying the reports that will be expected from a system the user implicitly states the scope of the required system. If the reports can be described in clear and precise may at the user's level then the basic data needs can be derived. User has familiarity with reporting and if an easy-to-use language with good description power is available then a specific start can be made towards information need identification.

Once an interactive report language processor is available it can be extended to help the user in formatting the reports and it can help the system analyser in interpreting user's needs in greater detail.

The report description method developed here has a very special advantage of not being a language in isolation but it identifies with the further development of a user level data and document description language.

It also has special features where reports which have to be related to the reporting medium or the reporting hardware can also be completely described.

The data selection feature can be extended. For example arithmetic expressions on data items may be used for selection within the basic structure. Ready made features may be increased or reduced to get ease of description without curtailing the power of description.

CHAPTER 4

DATA DESCRIPTION FOR INFORMATION SYSTEMS

There are many approaches to descriptions of information transformations required for the target information system. In the model building approach the user has to build up concepts about actions which constitute the real life system. The user, in general, may have to build up a very large number of concepts and for handling real life systems this method can become very cumbersom. In the questionnaire approach a generalised model is developed for the user and a supporting questionnaire, pertaining to the problem area, interacts with the user to make the necessary changes for adapting the model for the user. This method can be used for highly specialised applications. Also to assemble the system again, in the light of the change initiated by the user is not always possible. In this chapter a different approach is proposed for generalised information system descriptions.

To describe an information system it is not necessary to describe all the functions and the entities of the organisation. Information system has a specific role in the organization which is that of selective data handling. Generally a well structured description of the involved data is not available. A purely non-procedural description of data only in the form of a set of predicates or properties is very difficult. The user can, however, give a description of how he can get what he wants through repetitive selection of data. In other words,

the user can describe the required information system by giving one logical design or statement of data processing. Then, any description, which can give rise to equivalent data transformations can be used to describe the system and to generate technical designs of the system.

Information systems which are data processing oriented and do not necessarily have a library-like function deal with few different types of entities and a large number of functions related to such entities. Such functions give rise to abstract concepts as properties of these entities. For example SALES and COSTING, are concepts which are generated entities and are entities formed by data associations. Therefore instead of clear cut properties and concepts there are mainly data-associations to be described.

An additional problem exists due to large cardinality of data. Different occurrences cannot be provided with names. Therefore, instead of directly addressing data, data selection has to be used to limit the scope of names in a local sense. For example, data about STUDENTS with GRADE = 9 may be selected for the purpose of describing other properties for these STUDENTs.System does not maintain different names for selecting students with different grades.

4-1. STRUCTURE OF DATA TRANSFORMATION

In describing information transformations, there may be calculations, groupings, assignment or conditions. Data can be visualised

in the form of tables where rows represents the entities or concepts and columns represent the properties. For example, in conventional sequential systems there are fields and records. If there are a large number of entities and a small number of properties there are more rows and few columns. Entities are divided into classes and aggregate or overall properties of such classes are derived. For example, total quantity of order for a commodity may be derived by aggregating all the orders for the commodity. The divisions into such classes is done according to properties or the values of the columns. If a property can take a large number of values then such classes are again large in number and can not be identified by names. For example, if ORDER's have two properties such as TYPE and CUSTOMER-NAME of ORDER's. TYPE can take values "TEMPORARY" or "PERMANENT" and CUSTOMER-NAME can take any value from a set of (say) 100 names. Then based on TYPE, ORDER's can be named as TEMPORARY-ORDER's and PERMANENT-ORDER's. Based on CUSTOMER-NAME, no such names can be given even if name-wise sets have to be referred to. Therefore, it becomes necessary to have two type of features in an information system description language. One is that of subsetting the entity set into classes. The other is of being able to perform aggregation of information or of describing functions of subsets.

However, the functions of subsets are generally describable by using simple calculations. Aggregation of information is not like that of statistical processing. The difference is there because the

information developed by the information processing system is normally used by people at different levels in the organisation. Whereas, results of statistical processing are normally used by small groups of trained people.

4-2. DATA SELECTION AND SUBSETTING

Data selection and data subsetting are two features which have been incorporated in the language described here. Depending on the values of a property, a set may be divided into classes which are functionally similar. For each value of that property one instance of the class or the subset of data is generated. This is done by using "EACH" function. These classes have to be identified or made available for generation of aggregate information and for further data selection. For example, once the STUDENT's have been divided into classes according to GRADE, then value of GRADE can be used to identify the subset of students in the GRADE. GRADE becomes a subset property. This is achieved by using a block structure. A block is a sequence of procedures to be done on all sets which are input to the block, considering one set at a time. The procedures may be of the following type -

(1) Data subsetting procedures

(2) Data selection procedures

(3) Data generation procedures (data generation by calculations and assignment).

4-2.1 DATA SUBSETTING PROCEDURES

The basic tool towards the aggregation of procedures is provided by the data subsetting procedures. A subsetting procedure operates on a set of data. If the set is input to such a procedure the whole set becomes available to the procedure at a time (conceptually) though the procedure may operate on elements one by one. The procedure outputs a number of subsets each having some elements. It also attributes some properties to each subset. Each property has one value for one subset and may also be used to further identify the subset. Such subset properties then become global properties at subset level and in concept are no more associated with the elements of the subset on an individual basis.

To be able to operate on these subsets a number of any kinds of procedures&transformations may now be written in a block at a level lower than that of the subsetting procedure. For these procedures the subsets are made available (one subset at a time) along with all its elements. These procedures may use only subset properties and not individual element properties unless the subsetting has been done to an element level.

Examples:

```
FOR EACH ORDER-NO OF ORDER:S
FOR EACH OF ORDER:S
FOR EACH (ORDER-NO, ORDER-DATE) OF ORDER:S
```

The above three examples, subset the set of all ORDER's into classes of ORDER's, with same ORDER-NOs, having ORDER-NOs as the class property and so on.

4-2.2 DATA SELECTION PROCEDURES

Once a subset has been generated with some subset properties these subsets may be selected by describing conditions over set properties. A simple example is given in the following:

```
FOR ORDER:S BEGIN
    FOR EACH ORDER-NO OF ORDER:S BEGIN
        FOR ORDER-NO = 10 BEGIN NO-ACTION END
        FOR ORDER-NO NOT = 10 BEGIN
            COUNT ORDER'S AS NO-OF-ITEMS
            FOR EACH OF ORDER:S BEGIN
            MAKESHORT-QTY  = ORDERED-QTY - RECEIVED-QTY

                                    END
                                END
SUM SHORT-QTY OF ORDER:S AS REQUIRED-QTY
CREATE INVOICE (NO-OF-ITEMS, REQUIRED-QTY, ORDER-NO)
END
```

The sequence or procedures which constitute the block at the next level may operate only on those subsets which are selected by the procedure creating the block. From one class of subsets only one subset may be used by procedures. It is implied that procedures will be repeated on each subset of the class, separately, disregarding any sequence in which these subsets may be present. More complicated selection procedures may be described using the data and other classes of subsets that may be available for a procedure according to the context. The context is further described in Section 4-2.4.

4-2.3 DATA GENERATION PROCEDURES

There are various kinds of data generation procedures. Element oriented procedures have simple calculations, assignments and conditional assignments on different properties of elements. One element of class is available at a time for these procedures and one element may be generated at a time. An element may be a singleton element or a subset. Whatever is being treated as an element should be available as an element at the level of the procedure, i.e., a procedure has to be preceded by relevant subsetting procedures.

Other than the element oriented procedures there are set functions which operate on a subset. These may not take the subset as an element with only aggregate subset properties but may treat the subset as a set of elements. For built-in set functions like SUM, NUMBER, ALLOCATE, COUNT, MIN, MAX, FIRST, LAST, K-TH etc., the set is treated like a block box. User-written set functions are very often needed and Examples 4-1 describes one such case. In Chapter 3, Example 3-3 also describes one such case. Here, the subset is not treated as a black box, but the elements are created, ordered, and processed for aggregation at the element level as well as subset level to define such user-written set functions.

EXAMPLE 4-1

(Running profile of orders, receipts and amount for customers)

```
FOR CUSTOMER-ORDER:S BEGIN
    FOR EACH CUSTOMER-NO OF CUSTOMER-ORDER:S BEGIN

        MAKE DELIVERED-QUANTITY, ORDER-TO-DATE,AMOUNT,ORDER-SERIAL-
                                                          NO = 0
        FOR EACH OF CUSTOMER-ORDER:S, SEQUENCE DATE, ASCENDING BEGIN
            INCREASE ORDER-SERIAL-NO BY 1
            INCREASE ORDER-TO-DATE BY ORDERED-QUANTITY OF CUSTOMER-ORDER
            MULTIPLY RECEIVED-QUANTITY OF CUSTOMER-ORDER BY PRICE OF
                                              CUSTOMER-ORDER GIVING VALUE
            INCREASE AMOUNT BY VALUE
            CREATE CUSTOMER-ORDER-STATUS (ORDER-SERIAL-NO, ORDER-TO-DATE
                                                          AMOUNT;
                    DATE, CUSTOMER-NO = ALL CUSTOMER-ORDER)
            END
        MAKE FINAL-AMOUNT = AMOUNT
        SUM RECEIVED-QUANTITY OF CUSTOMER-ORDER AS DELIVERED-QUANTITY
        CREATE CUSTOMER-BILL (AMOUNT, DELIVERED-QUANTITY; CUSTOMER NO
                              = CUSTOMER-NO OF CUSTOMER-ORDER)
        END
    END
```

In the example given above, AMOUNT is an aggregate property for a subset of all the customer orders for one customer which is created by a user written procedure. ORDER-TO-DATE for running order status is an element property which is created by a user written set function (i.e., agreegating a property of earlier elements in the set).

In general, a set property at the higher level is always available for use alongwith the element properties at the lower level because the element belongsto the set. Therefore, what is a subset for a higher level is a set for a lower level description. If a set property is

updated at the element processing level, then at set level it implies a generation of a new property.

4-2.4 DATA NAMING AND BLOCK STRUCTURE

Data naming, which is based on a meaning for the name (in accordance with the context), makes use of the block structure. To begin with it is assumed that there is a database containing all the relevant data sets and their versions. The sets which are being considered for the data transformation, are selected. A particular version of these sets may also be selected. For example -

FOR ORDER:S, CUSTOMER:S, TRANSACTION:S

An s with a :, shows that a number of elements constitute the set. This set is available for processing as one subset containing the whole set with only one set property, (or the subset property) the property being its name. Therefore, at this level it is selected by explicitly giving the name. The very first procedure will naturally be a subsetting, because it is not attributed any properties at this level which are identifiable. At level one any subsetting procedure can take one set as input and a number of subsets or classes can be created as output. The number of classes generated may be only one or more. Suppose a function subsets one set into three, being that of ORDER:S. The meaning of ORDER:S will be altered as follows -

```
FOR ORDER:S, CUSTOMER:S BEGIN
    FOR EACH ORDER-NO of ORDER:S BEGIN
        FOR EACH CUSTOMER-TYPE OF CUSTOMER:S BEGIN
            Comment: meaning at this level
              CUSTOMER : A set of customers with a set property
                         customer-type, identifiable by type
              ORDER : A set of orders with a set property
                      order-No., identifiable by number.
              Any predicate or description at this level will be
              true for all such sets individually.
```

Procedures at the next level will be repeated for all such sets. This gives rise to a control structure for implied repetition for different sets. It may be observed that like we have PERFORM, DO, IF-THEN-ELSE, DO-WHILE, REPEAT and CASE etc., as control structures in programming languages, for controlling procedures, data transformation languages need data oriented control structures for two reasons -

1. Commonly used structures for ease of description;

2. Basic structures, to give the power of control.

Example 4-2 shows transformations involving a number of data sets and Example 4-3 shows a bills-of-material description to illustrate the update needs and capabilities. Normally the repetitions are due to data cardinality, but in Example 4-3 they are due to data-values.

Examples 4-2 and 4-3 are as follows:

EXAMPLE 4-2

```
FOR TRANSACTION:S, MASTER:S, ERROR-MASTER:S BEGIN
  FOR EACH ORDER-NO OF TRANSACTION:S BEGIN
    FOR MASTER (ORDER-NO = ORDER-NO OF TRANSACTION) BEGIN
      FOR MASTER PRESENT BEGIN
        ADD QUANTITY OF TRANSACTION TO QUANTITY OF MASTER
                                    GIVING NEW-QUANTITY
        UPDATE MASTER (QUANTITY = NEW-QUANTITY) END
        FOR CODE OF TRANSACTION NOT = CODE OF MASTER BEGIN
          FOR ERROR-MASTER (ORDER-NO = ORDER-NO of MASTER) BEGIN
            UPDATE MASTER (REASON = REASON OF ERROR-MASTER) END
OVER

        (One transaction per order-no is an integrity
                    constraints, here)
```

EXAMPLE 4-3

```
FOR PART:S BEGIN
  FOR PART-NO OF PART = 137 BEGIN
    CREATE SUB-PART (= ALL OF PART) END
OVER

FOR SUB-PART:S, PART:S, SUB-PART-NO:S BEGIN
  FOR EACH OF SUB-PART BEGIN
    FOR STATUS OF SUB-PART = "ELEMENTARY" BEGIN
      CREATE BASIC-PART (= ALL OF SUB-PART) END
    FOR STATUS OF SUB-PART = "ASSEMBLY" BEGIN
      FOR SUB-PART-NO (PARENT-PART-NO = PART-NO OF SUB-PART) BEGIN
        FOR EACH SUB-PART-NO BEGIN
          FOR PART (PART-NO = PARENT-PART-NO OF SUB-PART-NO) BEGIN
            ADD SUB-PART (= ALL OF PART)
              END
OVER
```

To show the compactness of description, some examples of data base queries are being taken in the following.

Query 3.11, [COD 73]

For each project, obtain as a triple, the project number, project name and supplier location, for all suppliers who supply that project.

```
RANGE PROJECT J
RANGE SUPPLIER S
RANGE SUPPLY Z SOME
GET W(J.J#, J.NAME, S.LOC):(J.J# = Z.J#) ∧ (Z.S# = S.S#)
FOR PROJECT:S, SUPPLIER:S, SUPPLY:S BEGIN
    FOR EACH PROJECT BEGIN
        FOR SUPPLY (J# = J# OF PROJECT) BEGIN
            FOR EACH SUPPLY BEGIN
                FOR S# OF SUPPLIER = S# OF SUPPLY BEGIN
                    CREATE W(J#, NAME = ALL OF PROJECT; LOC =
                                        LOC  SUPPLIER END
OVER
```

Problem

Query 3.10 [COD 71]

Give names and locations of all suppliers, each of whom supplies all projects.

```
RANGE SUPPLIER S
RANGE PROJECT J ALL
RANGE SUPPLY Z SOME
GET W(S.NAME, S.LOC) : (S.S#= Z.S#) ∧ (Z.J# = J.J#)
FOR PROJECT:S, SUPPLIER:S, SUPPLY:S
    COUNT PROJECT:S AS NO-OF-PROJECTS
    FOR EACH OF SUPPLIER:S BEGIN
        FOR SUPPLY (S# = S# OF SUPPLIER) BEGIN
            COUNT SUPPLY:S AS NO-SUPPLIED
            FOR NO-SUPPLIED = NO-OF-PROJECTS BEGIN
                CREATE W(NAME, LOC = ALL OF SUPPLIER) END
OVER
```

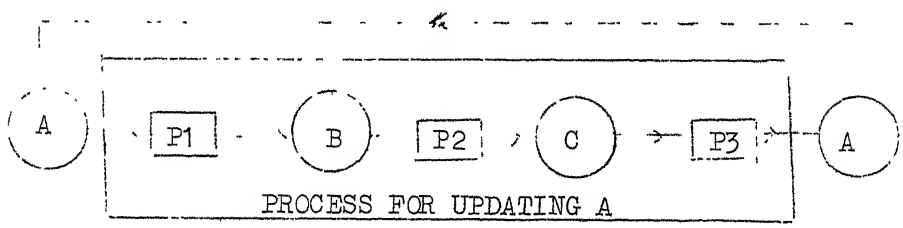
It can be easily shown that operations of union, itersection, difference, projection and restriction, as defined in the relational algebra can be easily described using this approach.

4-3. CONCEPT OF UPDATES IN DATA PROCESSING

In a general sense a data processing system constantly updates and processes data. Updating is a procedural concept. In an attempt to make system descriptions more declarative and less procedural orientation of the language should be such that update is used as little as possible. The situations under which an update is used, can be visualised as follows:

1. Elementary data item updates
2. Entity set updates
3. Property updates for entities
4. Data set updates

The essential type of update is that of change in the value of the properties of a real life entity like customer or a commodity, about which the information is kept. Other kinds of updates found in data transformations are due to time cycles, data cardinality or cycles of processing. In general, an update cycle may be present in a data transformation description as given in the figure below.



Such a description does not make clear why the process has been described as an update. It just describes that P1, P2 and P3 are three processes operating on A,B and C and they together form a cycle. In an attempt to bring out the need or the reason for update, so that it can be utilised for further analysis and design, different constructs are used for different kinds of updates.

Elementary data item updates, like QUANTITY QUANTITY + AMENDMENT are used in data transformation descriptions. These are basically required to construct n-ary operations (an operation involving n data items) using binary arithmetic operator. For example, BILL TOTAL = ORDER 1 + ORDER 2 uses a binary addition operators. For seven orders, a ready made n-ary operator SUM can be used SUM ORDERS AS BILL-TOTAL. We could also have BILL-TOTAL = ORDER-QUANTITY * RATE-OF-ORDER. In this case a BILL-TOTAL for seven ORDERs will be obtained by updating it for every order, such as, BILL-TOTAL ← BILL-TOTAL + ORDER-QUANTITY * RATE-OF-ORDER. Such updates are described by using constructs like ADD AMENDMENT TO QUANTITY, INCREASE QUANTITY BY AMENDMENT etc. Some n-ary constructs like SUM, COUNT, NUMBER etc., are also defined, to maintain the ease of use.

Entity set updates, for example, updated customer set after deleting selected customers, are handled by ADD and DELETE. This avoids update at a description level and makes it, a purely data processing level function. Data set updates for removing obsolete data from data sets and adding new data to data sets (for example keeping data about weekly orders) are also made implied data processing functions.

Updates for properties of entities and addition and deletion of entities have to be given by the user. Data processing oriented updates are not a part of the description. In this way, descriptions become less procedural and user description remains at a system level. This helps in understanding the updates and their needs.

4-4. TIME AS A SYSTEM CONCEPT

The data processing systems being considered here are batched or clocked systems and not real time systems. In such systems, sets of data are collected over preconceived time intervals. Functionally similar data, occuring at different points of time, may be identified by the associating a dimension of time with data. Time is required as an integral system concept for the following reasons:

1. All systems have, as a natural necessity, functions linked with some or the other time aspect. Any description of the system cannot avoid referring to time.

2. Time may be defined as data type in the system because it has a very standard usage and a very complicated data type having lot of sub-types etc., and complicated naming conventions. If the user has to describe the data type in the course of the description then a lot of information about the structure of the system gets disintegrated and remains no more available for the purposes of analysis and design.

For example, suppose there are ORDERs for each day of the week. There is a data item DAY-OF-ORDER. A data transformation description may compare DAY-OF-ORDER with the DATE-OF-RUN giving a detailed matching

procedure, extracting the DAY from the DATE and then comparing the two data items. If a facility is given for directly using the DAY of a DATE or for declaring DAY-OF-ORDER as a special data item which identifies the weekly ORDER's with the respective days then this information can be used to analyse the data and structure it.

3. The storage of data in the system depends on its life time and the times at which it is required. For example, tomorrow's invoices may have to be printed today. In other words, data processing has to be synchronised with real time.

4. Data processing and the real environment synchronises by using time as a data item. For example, orders of the month of March are identified by the date of order.

As a system concept, conversion from a time data item to real time identification and vice-versa, are needed. A case study given in Chapter 6 has an example where an amendment document has two time identifications. Amendment entering on Monday, as the amendment for Tuesday. In such cases at a time, one of the time items appearsas a data item and the other as a time tag. This is achieved by two features of the language -

1. PERIOD may be used to declare the time tag

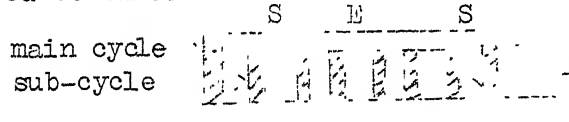
2. Data item may be equated to a time-item.

Time as a system concept, also needs to allow additional time intervals, periods and sub-intervals which may be very specific to a system. For example four shifts a day is a specific interval.

Saturday to Friday week is a specific interval whereas Monday to Sunday week is a general interval.

Time as a system concept, should allow making a reference to any point in time. For example, two days before today. For doing this, the current real time status should be available for reference. For example, one may like to check if the week of the order is current week.

The user is clearly able to perceive timings of various operations in the system and the system designer is able to generate data items related to time.



From the view point of computerised processing, a time interval may be divided into three parts. Beginning of the interval, when the system processing can be done. Middle of the interval during which things belonging to that interval, may happen in the environment. End of the interval during which system processing can be completed. For example, for the day, the order's of the day have to be processed. It implies processing at the end of the day. Normally end-of-the-interval processing may be taken as usual and the beginning-of-the-day processing may be considered only if there is a special declaration. If the distribution of the processing is specifically complicated over different time intervals and sub-intervals of time then the block structure of description can be readily exploited.

4-5. DESCRIPTIONS FOR INPUT DOCUMENTS

Essential features required for the description of documents or the inputs to the system are similar to those for report descriptions. The main difference is that reports are more structured than input documents. The input from the environment is prepared by people, and hence the data is more raw, or less processed and therefore it is in simple forms. It may have more codes built into the data items to avoid data entry errors. Input forms may have unfilled portions, which are allowed to be optional and in that case they should not be used to generate data item strings. This occurs becauses forms have a rigid structure and do not adjust themselves like reports, depending on the presence of data. This needs special feature in being able to ignore unfilled portions.

Input media control procedures like skipping to a new form and input media condition sensing like creating a code from the line number on the form, etc., are similar to corresponding report description features. VALUE clause for data items may be used to describe data integrity constraints on individual data items. In some cases the input medium may be linked to a message control system. Then input may have more report-like structure according to the presence or the absence of data.

The case study given in the Chapter 6 illustrates these features.

4-6. SYNTAX OF THE LANGUAGE

The syntax of the language is given below. Report description part of the language, which is given in Chapter III is being omitted here. Syntax for input descriptions is same as that for report descriptions and is not being repeated.

<APPLICATION-SYSTEM> ::= <DESCRIPTION> END-OF-DESCRIPTION

<DESCRIPTION> ::= <SINGLE-MODULE> | <DESCRIPTION> <SINGLE-MODULE>

<SINGLE-MODULE> ::= <TIME-MODULE> | <OTHER-MODULE>

<OTHER-MODULE> ::= <NECESSARY-CONDITION> <ANY-MODULE>

<ANY-MODULE> ::= <REPORT-MODULE> | <DOCUMENT-MODULE> |
<DATA-TRANSFORMATION-MODULE>

<REPORT-MODULE> ::= REPORT <REPORT-DESCRIPTION>

<DOCUMENT-MODULE> ::= <DOCUMENT-DESCRIPTION>

<NECESSARY-CONDITION> ::= <EMPTY> | FOR <CONDITION-DETAIL>

<CONDITION-DETAIL> ::= EACH <RELEVANT-DATA-ITEM>
WHEN <SINGLE-ITEM-CONDITION>

<RELEVANT-DATA-ITEM> ::= <DATA-ITEM> | <SYSTEM-DATA-ITEM>

<SYSTEM-DATA-ITEM> ::= <TIME-CYCLE> | <TIME-INSTANT>
BEGINNING OF <TIME-CYCLE>

<SINGLE-ITEM-CONDITION> ::= <DATA-ITEM> <RELATION> <DATA-ITEM>

<DOCUMENT-DESCRIPTION> ::= DOCUMENT <DOCUMENT-NAME> TO DATA SET
<DATA-SET-NAME> <DOCUMENT-DETAIL> OVER

COMMENT: The document description is identical to report description given in Chapter III other than the following additions.

<RANGE-OF-VALUE> ::= <RANGE-OF-VALUES> RANGE-OF-VALUES | < LITERAL > |
<LITERAL TO LITERAL>
SIMPLE-PROCEDURE ::= UNFILLED, IGNORE | < INTEGRITY-CONSTRAINT >

< VALUE-CLAUSE > ::= VALUE (RANGE-OF-VALUES)

<INTEGRITY-CONSTRAINT> ::= ALL REQUIRED

<COPY-STATEMENT> ::= SAME-AS <MODULE-NAME> |
SAME-AS <MODULE-NAME> USING
(<SET-OF-PARAMETERS>)

<SET-OF-PARAMETERS> ::= <PARAMETERS><SET-OF-PARAMETERS> |

<PARAMETERS>

<PARAMETER> ::= < SET-OF-WORDS> FOR <SET-OF-WORDS>

<SET-OF-WORDS> ::= <WORD> | <WORD> < SET-OF-WORDS >

<DATA-TRANSFORMATION-MODULE> ::= DATA-DESCRIPTION
< MODULE-NAME >
< DATA-DESCRIPTION >
OVER

< MODULE-NAME> ::=< VARIABLE-NAME | NIL <EMPTY>

< DATA-DESCRIPTION> ::= <DATA-DESCRIPTION>< DATA-DESCRIPTION> |
FOR< DATA-CONDITION> BEGIN
<DATA-DESCRPTION > END | <DATA-STATEMENT >

< DATA-CONDITION> ::= <DATA-SELECTION> | <DATA-SUBSETTING>

< DATA-STATEMENT> ::= <ENTITY-STATEMENT > |
<SET-PROPERTY-GENERATION > |
<DATA-ELEMENT-GENERATION > |
<OTHER-STATEMENTS >

< ENTITY-STATEMENT> ::=< CREATE-STATEMENT> |
< ENTITY-SET-UPDATE> |
< ENTITY-PROPERTY-UPDATE >

< CREATE-STATEMENT> ::= CREATE< SET-NAME> (<SET-DOMAINS>)

< ENTITY-SET-UPDATE > ::= ADD< SET-NAME> (<SET-DOMAINS>) |
DELETE<SET-NAME> (<SET-DOMAINS>)

< ENTITY-PROPERTY-UPDATE > ::= UPDATE< SET-NAME> (<SET-DOMAINS>)

< DATA-ELEMENT-GENERATION > ::= MAKE <VARIABLE-NAME> = <ANY-DATA> |
INCREASE <VARIABLE-NAME> BY <ANY-DATA> |
<MULTIPLY-CLAUSE > | OTHER-CALCULATION >

< ANY-DATA > ::= <DATA-ITEM > | <LITERAL > | <TIME-INSTANT >

< SET-PROPERTY-GENERATION > ::= SUM <DATA-ITEM > AS <VARIABLE-NAME> |
COUNT <DATA-SET > AS < VARIABLE-NAME> |
NUMBER ON <DATA-ITEM > SEQUENCE > AS
<VARIABLE-NAME >

< SET-DOMAINS > ::= <VARIABLE-NAME-STRING > = ALL OF < SET-NAME > |
ALL OF <SET-NAME > |
< VARIABLE-NAME> = <DATA-ITEM > |
< VARIABLE-NAME-STRING> ; |
< SET-DOMAINS > , <SET-DOMAINS >

<SEQUENCE > ::= ASCENDING | DESCENDING

<DATA-ITEM> ::=< VARIABLE-NAME > | <VARIABLE-NAME> OF <DATA-SET > |
<DATA-ITEM >REDESCRIBED BEGIN <DATA-ITEM-STRING > END |
POSITION <INTEGER > TO <INTEGER > OF <DATA-ITEM >
<SET-NAME > ::= <DATA-SET-NAME> < LOCAL-RENAME > | < VARIABLE > : S | < VARIABLE >
<DATA-SET-NAME > ::= < DATA-SET > | < DATA-SET > OF <TIME INSTANT >

<VARIABLE-NAME-STRING> ::= <VARIABLE-NAME> | VARIABLE-NAME VARIABLE-NAME-
STRING>
<DATA-ITEM-STRING> ::= < VARIABLE-NAME-STRING> | <VARIABLE-NAME> REPEATED
< INTEGER > TIMES

< DATA-SELECTION > ::= <VALUE-SELECTION > | < SET-STRING >
<SET-CHECKING >

<SET-CHECKING > ::= < SET-NAME> <CHECK >

< CHECK > ::= PRESENT | ABSENT | TYPE < SET-NAME >

< VALUE-SELECTION > ::=< DATA-ITEM> RELATION GIVEN-ITEM >

< GIVEN-ITEM > ::= <DATA-ITEM> | <LITERAL > | < TIME-DATA-ITEM >

< SET-STRING > ::= <SET-NAME> | < SET-NAME> < SET-STRING >

< SEQUENCING > ::= SEQUENCE ON < SIMPLE-DI-STRING > < SEQUENCE >

< DATA-SUBSETTING> ::= <SUBSETTING> <SEQUENCING>

< SUBSETTING > ::= <TWO-SET-SELECTION > | <INTEGER > TIMES
EACH <SIMPLE-DI-STRING> | EACH < TIME-PERIOD>
< SELECTED-DI-STRING > |
< TIME-TAG> OF < DATA-SET >
< DATA-SET> TYPE <DATA-SET >

< TWO-SET-SELECTION >::=< SET-NAME>(<SET-DOMAINS>)

<SELECTED-DI-STRING > ::= <DI-SELECTION>|< DI-SELECTION> <CONNECTOR >
<SELECTED-DI-STRING>

<CONNECTOR > ::= ,|AND |OR

<DI-SELECTION> ::= <SIMPLE-DI-SELECTION > |< RANGE-DI-SELECTION >

<SIMPLE-DI-SELECTION >::= <DATA-ITEM>= <ANY-ITEM >

<RANGE-DI-SELECTION > ::= < DATA-ITEM > = <VALUE-STRING >

<VALUE-STRING > ::= <ANY-ITEM>| <VALUE-STRING>, <VALUE-STRING >
<ANT-ITEM >TO <ANY ITEM >

<GLOBAL-RENAME > ::= <DATA-SET-NAME > DEFINED AS <DATA-SET-NAME >

< OTHER-STATEMENTS > ::= <COPY-STATEMENT>|<GLOBAL-RENAME > |
<LOCAL-RENAME > | < VERSION-TIME >

<LOCAL-RENAME > ::= < DATA-SET-NAME > CALLED < DATA-SET-NAME >

< VERSION-TIME > ::= PERIOD <FORMATTED-TIME-INSTANT > IS < GIVEN-ITEM >
OF <SET-NAME >

< TIME-FORMAT > ::= < SUB-CYCLE> OF <SUPER-CYCLE >BY <NUMBER > |
< CYCLE> BY <NUMBER >|< CYCLE > BY <NAME >|
< TIME-FORMAT > , < TIME-FORMAT >

< FORMATTED-TIME-INSTANT > ::= < TIME-INSTANT > (<TIME-FORMAT >)

< CYCLE >::= DAY | WEEK | YEAR | MONTH | QUARTER | HOUR < USER-GIVEN-CYCLE>

<USER-GIVEN-CYCLE >::=< SUPER-CYCLE >|< SUB-CYCLE >

< SUPER-CYCLE >::= < SUPER-CYCLE-DEFINITION >

< SUB-CYCLE > ::= PERIOD <INTEGER > < VARIABLE-NAME> IN < VARIABLE-NAME>

< TIME-DATA-ITEM >::= <TIME-INSTANT >| <FORMATTED-TIME-INSTANT >

<TIME-INSTANT . > ::= TODAY | TOMORROW | YESTERDAY |
< USER-CONSTRUCTED-INSTANTS > |
< SYSTEM-CONSTRUCTED-INSTANTS > |
< USER-DEFINED-INSTANT >

< USER-CONSTRUCTED-INSTANTS> ::= < CYCLE-COUNT > < CYCLE >
< TIME-RELATION ><TIME-INSTANT >

< USER-DEFINED-INSTANT > ::= PERIOD < VARIABLE-NAME >FROM <TIME-INSTANT > TO
< CYCLE-COUNT > ::=< EMPTY>|< INTEGER > <TIME-INSTANT >

<TIME-RELATION > ::= BEFORE | AFTER

< SYSTEM-CONSTRUCTED-INSTANTS > ::= CURRENT <CYCLE > |
LAST< CYCLE > |
NEXT <CYCLE>

< SUPER-CYCLE-DEFINITION > ::= < OVER-LAPPED-CYCLE - DEFINITION> |
<EVEN-CYCLE-DEFINITION > |
<UNEVEN-CYCLE DEFINITION >

< OVERLAPPED-CYCLE-DEFINITION > ::= PERIOD <USER-CYCLE >
IS FROM <TIME-INSTANT> TO <TIME-INSTANT>

< EVEN-CYCLE-DEFINITION> ::= < EVEN-CYCLE >

< EVEN-CYCLE> ::= PERIOD< USER-CYCLE > IS MADE OF < INTEGER >< CYCLE >

< UNEVEN-CYCLE-DEFINITION >::= <UNEVEN-CYCLE >| < UNEVEN-CYCLE>
< UNEVEN-CYCLE-DEFINITION >

< UNEVEN-CYCLE >::= PERIOD< USER-CYCLE> (<INTEGER >)
IS MADE OF <INTEGER>< CYCLE >

< USER-CYCLE > ::= <VARIABLE-NAME >

< EVEN-CYCLE-NAME> ::= < NAME-CLAUSE >| NAME-CLAUSE ><EVEN-CYCLE-NAME >

< NAME-CLAUSE> ::= NAME OF< CYCLE> (<INTEGER >) IS <LITERAL >

4-7. CONCLUSION

The system description approach taken in the language described in this chapter preserves most of the structure of user's system in a usable or understandable form. This is achieved firstly by using time as a system concept. By eliminating explicit use of loop-structure by providing features for data cardinality and different kinds of update requirements.

To give completeness to the formal system description, inputs can also be described using the DOCUMENT feature. Timing needs of the system and its relation with the environment, can be integrated with

the system description. A limitation of this approach is that it does not aim at getting the informational view of the system or understanding the system but it only aims as getting a precise data model of the system and depends on the user to be able to state his information requirements in terms of data requirements. In this way, instead of making the description fully declarative, it remains procedural to the extent that the 'know how' of data associations for the required information has to be given by the user.

At the detail level, the different kinds of data mappings may be obtained by the mechanisms of subsetting, data selection and data generation. Built-in mechanism for 1 → M subsetting, 1 → M data selection and M → 1 (COUNT,SUM), 1 → 1 (REPORT data items), 1 → M (NUMBER) data generation have been provided. Here 1 → M etc use '1' in a general sense of either a set or an element. However, M → N mapping on all the three mechanism, in a user described, explicit manner is facilitated by using the block structure and the related subset naming mechanism which is implied by the structure. A limitation at this level, is that the subsetting procedures can put one element of the set, only in one of the subsets at a time. If multiple copies of an element have to be put in different subsets then this can be indirectly done by selecting more copies at the beginning of a procedure. However, this limitation also limits the user from making unreasonable descriptions while being unaware of it.

In the end, it can be said that the approach leads to a simple description, concisely, so that the description can be used for computer-aided analysis and design. It is structured to maintain the structural properties of the described system at a usable level. An information system links itself with reality and the associated organisations or the firm by inputs, outputs and the associated timings. These facilities are provided in this approach for completeness of description.

CHAPTER 5

SYSTEMS ANALYSIS

In the last two chapters of the thesis a system description language, SDL, has been developed which can be used to fully describe an information processing system required by the user. The inputs, outputs and the timing descriptions formulate the interaction of the information processing system with its environment or the organisation which will use the system. The data transformation descriptions formulate how the system should behave, or should work out the data associations.

Such a description can be used as a documentation aid to the systems analyst. It will, however, be of much greater impact if it can be used as a design aid. In this chapter we will discuss how such a description can be used for computer aided system analysis and design. Systems analysis, today, is done manually by a systems analyst. In the following section, we will discuss how simple analysis about the availability of the required data can be done. Then we will proceed to discuss why such an analysis is much less than what a systems analyst does. In Sections 5-1 to 5-5 we will discuss some new methods of systems analysis which exploit the structure of SDL, for interactively extracting information relevant to analysis and design.

Let us start with a small example of a system which keeps master information about the costs of products. This system receives invoices and checks certain items like discount and product value. It is an invoice audit. It prints out a report for those invoices where some discrepancies are found. The description of reports, inputs and data transformations are given in SDL in the following. (The format diagrams of input forms and reports are given purely to assist the reader.)

PRODUCT FORM

(10 Lines per form)

PRODUCT CODE	STANDARD LABOUR PRICE	STANDARD MATERIAL PRICE	MAXIMUM DISCOUNT PERCENT	DESCRIP-TION	SELLING PRICE
1345	97.34	13.56	12	TYPE-1	153.67
....	.....	.....	..	.....	.....
....	.....	.....	..	.....	
....	.....	.....	..		
....	.....	.....			
....	.....				
....					

```
FOR EACH DAY
DOCUMENT PRODUCT-FORM TO DATA SET COST-MASTER:S
FOR EACH 10 COST-MASTER:S BEGIN
    NEW-FORM AFTER 5 LINES
    FOR EACH COST-MASTER BEGIN
        PRODUCT-CODE, STANDARD-LABOUR-PRICE, STANDARD-MATERIAL-PRICE
        SELLING-PRICE DESCRIPTION MAX-DISCOUNT-PERCENT END
FORMAT PRODUCT-CODE FROM 4 SIZE 4 STANDARD-LABOUR-PRICE
FROM 11 PICTURE 99.99 STANDARD-MATERIAL-PRICE FROM 19 PICTURE
99.99 MAXIMUM-DISCOUNT-PERCENT FROM 28 NUMBER SIZE 2 DESCRIPTION
FROM 32 SIZE 10 SELLING-PRICE FROM 45 PICTURE 999.99
OVER
```

INVOICE FORM
PRODUCT CODE 1345 INVOICE-NUMBER 1001 YEAR XX WEEK-NO XX
CLASS OF TRADE XXX CUSTOMER NOS 1056 AREA LC QUANTITY 50
PRODUCT VALUE XXXX
DISCOUNT VALUE XXXX SIGNATURE STAMP

```
FOR EACH DAY
DOCUMENT INVOICE-FORM TO DATA SET INVOICE:S
FOR EACH INVOICE BEGIN
    NEW-FORM AFTER 1 LINE, PRODUCT-CODE, INVOICE-NUMBER, WEEK-NO
    YEAR-NO NEW-LINE CLASS-OF-TRADE CUSTOMER-NO, AREA
    QUANTITY NEW LINE PRODUCT VALUE
    NEW-LINE DISCOUNT-VALUE END
FORMAT PRODUCT-CODE FROM 15 NUMBER 4 INVOICE-NUMBER FROM 34
NUMBER 4 YEAR-NO FROM 44 SIZE 2 WEEK-NO FROM 52 SIZE 2
PRODUCT-VALUE EACH 15 SIZE 4 DISCOUNT VALUE FROM 15 SIZE 4
CLASS OF TRADE FROM 15 SIZE 4 AREA FROM 44 SIZE 2
QUANTITY FROM 52 NUMBER 2 CUSTOMER-NO FROM 34 SIZE 4
OVER
```

```
                    REPORT ON INVOICE EDITING                    PAGE 1
PRODUCT   INVOICE  CALCULATED  PRODUCT  MAXIMUM   DISCOUNT   COMMENT
 CODE       NO       VALUE     VALUE  DISCOUNT     VALUE

 0513       10        53.5      53.0                        COMMENT-1
 0600       1                                10        12   COMMENT-2
 0713       3         60.0      65.0         9        15   COMMENT-1
                                                           COMMENT-2
 0613       6         23.5      23.0                        COMMENT-1
 ....       ..        ....      ....        ..        ..   ..........
 ....       ..        ....      ....
```

```
FOR EACH DAY REPORT INVOICE-EDIT
FROM DATA SET INVOICE-AUDIT:S
BLANK 12, "REPORT ON INVOICE EDITING," PAGE-NUMBER
NEW-LINE, HEADING-LINE-1, NEW-LINE, HEADING-LINE-2
FOR EACH OF THE INVOICE-AUDIT BEGIN
  FOR DISCOUNT-ABOVE-MAX = 1 OR VALUE-DISCREPANCY-COUNT = 1 BEGIN
    NEW-LINE, PRODUCT-CODE, INVOICE-NO
    FOR VALUE-DISCREPANCY-COUNT = 1 BEGIN
      CALCULATED-VALUE, PRODUCT-VALUE END
    FOR DISCOUNT-ABOVE-MAX = 1 BEGIN
      MAXIMUM-DISCOUNT, DISCOUNT-VALUE END
    COMMENT-1
    FOR COMMENT-1 = BLANK BEGIN COMMENT-2, END
    FOR COMMENT-1 ≠ BLANK AND COMMENT-2 ≠ BLANK BEGIN
      NEW LINE, COMMENT-2 END
  FOR NEW-PAGE BEGIN PAGE-NUMBER END
FORMAT HEADING-LINE-1 FROM 1 VALUE "PRODU"-
-"CT INVOICE CALCULATED PRODUCT MAXIMUM DISCOUNT COMMENT"
HEADING-2 FROM 2 VALUE "CODE        NO."-
-"      VALUE   VALUE DISCOUNT VALUE"
PRODUCT-CODE FROM 2  SIZE 4  INVOICE-NO FROM 12  SIZE 2
CALCULATED-VALUE FROM 24 PICTURE 99.9  PRODUCT-VALUE FROM 36 PICTURE 99.9
MAXIMUM-DISCOUNT FROM 45 SIZE 2  DISCOUNT-VALUE FROM 55 SIZE 2
COMMENT FROM 60 SIZE 10
OVER
```

```
FOR EACH DAY
DATA-DESCRIPTION EDITING
FOR INVOICE:S COST-MASTER:S BEGIN
    FOR EACH INVOICE BEGIN
        FOR YEAR-NO = CURRENT YEAR BEGIN
            FOR COST-MASTER (PRODUCT-CODE = PRODUCT-CODE OF INVOICE) BEGIN
                CALCULATED-VALUE = SELLING-PRICE * QUANTITY
                FOR CALCULATED-VALUE ≠ PRODUCT-VALUE BEGIN
                    MAKE VALUE-DISCREPANCY = "1"
                    MAKE COMMENT-1 = "VALUE-DISCREPANCY" END
                MAX-DISCOUNT-VALUE = CALCULATED-VALUE * MAX-DISCOUNT-PERCENT
                FOR MAX-DISCOUNT-VALUE <DISCOUNT-VALUE BEGIN
                    MAKE DISCOUNT-ABOVE-MAX = "1"
                    MAKE COMMENT-2 = "DISCOUNT ABOVE MAX" END
                CREATE INVOICE-AUDIT (COMMENT-1,COMMENT-2,CALCULATED-VALUE,
                PRODUCT-VALUE MAXIMUM-DISCOUNT, DISCOUNT-VALUE
                PRODUCT-CODE INVOICE-NUMBER QUANTITY DISCOUNT-ABOVE-MAX
                VALUE-DISCREPANCY-COUNT)
OVER
```

Two inputs, namely PRODUCT FORM and INVOICE FORM, one report INVOICE AUDIT and EDITING data transformation, are used to describe the system. To analyse the system data needs for the reports and data availability from the input documents will be derived. Also, some data may not be available directly from the documents but from data transformations. Data transformation may need some data to carry out the transformation, and that should also be available in the system. In the following description the data will be represented as relations, for each of reference. Every data item will be given a composite name as 'data-item of set-name'. Every relation (as used here) will have 4 constituents : a relation name, an associated set name, a set of key data items, a set of attribute data-items.

A procedure to derive the data needs from reports has already been given in Chapter III. Similar procedure can be written for getting data availability from the documents. A similar procedure is given below to get the data needs and data availability for the data transformations.

1. Identify the data items with only one of the data sets used in the data transformation module, if there are more than one data sets. If two or more data sets have same data item name then the user should be asked to clear the error. In the above example, in the EDITING module, QUANTITY will be associated with INVOICE and SELLING PRICE with COST-MASTER. If there was a SELLING-PRICE in the invoice also then either the description should be CALCULATED-VALUE = QUANTITY * SELLING-PRICE OF COST-MASTER or the user will have to clear the doubt. For left hand side items of MAKE and the calculations, a set name should be generated and associated.

2. Associate a level-number with every data item, by starting with level-No. 1 and increasing the level-No. by 1 with every BEGIN and decreasing the level-No. by 1 with every END. The data-items should be entered in the sequence in which they are encountered, to maintain the block structure of the description.

3. For each statement of the type FOR EACH data set , introduce a special data item as IDENTIFICATION-NO at the proper place in the table, with the proper level No.

4. Mark all the data items on the left hand side of a calculation, MAKE and CREATE as available type. All those on the right hand side are to be marked as 'N' or needed items.

5. Mark all the items in FOR statements, excluding those on the right hand sides, as KEYS or 'K'. Others are to be marked as ATTRIBUTES or 'A's. Use FOR statements with equals using two data items, to list the synonyms.

6. Repeat the following steps for all the attribute type of items, to generate relations. Also repeat for the lowest key of relations of the type 'A' or available.

7. Take the level No. of the data-item. Reduce the level by one. Put all the key items of the first occurrence of this level as the keys in the relation being generated.

8. Repeat Step 7 till the level-No. becomes zero.

9. If the attribute is not a generated item, then keys belonging to its own set name are to be retained only.

10. All the relations which are identical but for the attribute data name may be consolidated in one relation by putting the attributes together as a set of attributes.

A list of relations, as worked out by applying the above procedure is given below. Codes are introduced for readability, within the parantheses.

Need or qualibility	Relation	Set-name	Keys	Attributes
(from report INVOICE-EDIT)				
N	R1	INVOICE-AUDIT (S1)	IDENTIFICATION-NO (S1 ID) DISCOUNT-ABOVE-MAX (S1F1) VALUE-DISCREPANCY-COUNT (S1F2)	PRODUCT-CODE (S1F3) INVOICE-NO (S1F4)
N	R2	S1	S1ID, S1F1, S1F2	CALCULATED-VALUE (S1F5) PRODUCT-VALUE (S1F6)
N	R3	S1	S1ID, S1F1, S1F2	MAXIMUM-DISCOUNT (S1F7) DISCOUNT-VALUE (S1F8)
N	R4	S1	S1ID, S1F1, S1F2	COMMENT1 (S1F9)
N	R5	S1	S1ID, S1F1, S1F2, S1F9	COMMENT-2 (S1F10)
N	R6	S1	S1ID, S1F1, S1F2, S1F9, S1F10	S1F10
(from document PRODUCT FORM)				
A	R7	COST-MASTER (S2)	IDENTIFICATION (S2ID)	PRODUCT-CODE (S2F1) STANDARD-LABOUR-PRICE (S2F2) STANDARD-MATERIAL-VALUE (S2F3) SELLING-PRICE (S2F4) DESCRIPTION (S2F5) MAX-DISCOUNT-PERCENT (S2F6)

Need or availability	Relation	Set Name	Keys	Attributes
(from document INVOICE FORM)				
A	R8	INVOICES (S3)	IDENTIFICATION (S3ID)	(S3F1) PRODUCT-CODE (S3F2) INVOICE-NUMBER (S3F3) YEAR-NO (S3F4) CUSTOMER-NO (S3F6) AREA (S3F7) QUANTITY (S3F8) PRODUCT-VALUE (S3F9) DISCOUNT-VALUE (S3S10) WEEK-NO
(from data transformation EDITING)				
N	R9	S3	S3ID	S3F2, S3F7, S3F1
N	R10	S2	S2F1	S2F4
A	R11	NIL (S4)	S3F3, S2F1 S3ID	CALCULATED-VALUE (S4F1)
N	R12	S4	S3F3, S2F1, S3ID	S4F1
A	R13	NIL (S5)	S2F1, S3ID	VALUE-DISCREPANCY (S5F1) COMMENT-1 (S5F2)
A	R14	NIL (S6)	S3F3, S2F1, S3ID	MAXIMUM-DISCOUNT-VALUE (S6F1)
N	R15	S4	S3F3, S2F1, S3ID	S4F1
N	R16	S2	S2F1	MAX-DISCOUNT-PERCENT (S2F6)

Need or availability	Relation	Set Name	Keys	Attributes
N	R17	S3	S3ID, S3F3	S3F9
N	R18	S6	S2F1, S3ID, S3F3	S6F1
A	R19	S7	S2F1, S3ID, S3F3	DISCOUNT-ABOVE-MAX (S7F1) COMMENT-2 (S7F2)
A	R20	S1		QUANTITY (S1F11) S1F1, S1F2, S1F3, S1F4, S1F5, S1F6, S1F7, S1F8, S1F9, S1F10
N	R21	S3	S3F3, S3ID	S3F8

Matching of the relations can be shown as follows:

R1, R2, R3, R4, R5, R6 ← R 20 R 18 ← R14

R17, R9 ← R8 R12 ← R8, R11

R16, R10 ← R7 R15 ← R11

Superfluous data: S1F11, S2F5

The above kind of matching is simple data analysis. There may be loops of data transformation in the definition. For this, further analysis may be done to identify errors in the definition. The above kind of analysis is present in the software systems which do computer-aided analysis. However, a systems analyst normally would have done the following analysis to find some more shortcomings of the above description.

1. This system description can have two interpretations. Product forms for all the products may be required every day or product forms for only the new products may be accepted everyday and may be used alongwith the old product information. If needs to be made clear, as to which of these interpretations is correct. User may assume that it is a well known conventional fact.

2. If a master record is absent then also the invoice should be marked for audit.

3. If the year is not correct then the invoice should be marked for audit.

4. If invoices are read product-wise, it may involve less reading of information.

5. If there are two products in the same product code then something should be done, unless it is an integrity constraint for the system.

This brings out some of the other aspects of real life system analysis. The structure and the level of the language developed in this thesis, has been selected in such a way as to facilitate more detailed systems analysis. In the following sections, we will discuss how the system descriptions may be refined, by interacting with the user in relation to various aspects of design and analysis, like integrity, selection, subsetting and cardinality.

5-1. DATA INTEGRITY AND SUBSETTING

At any level of processing only aggregate properties of the subsets may be used for data selection and generation. If the subset itself is an element, then all the element properties may be used for such a purpose.

To explain this by continuing the same example let the invoice audit system be expanded to keep product master information for different branches of the same product, using only one product code for one product. In this example the situation is very clear and the user may account for it. But in many cases omissions are made. In this situation the user should either put it as an integrity constraint or a necessary requirement from the data that there will be one cost-master for one product-code or user should do further data selection to identify the product with proper selling price and then use the information. The following procedure may be used to identify such situations and generating a query for the user.

1. Do the following for each data description module for subsetting for each data set used, taking the sets one by one.

2. For the set, create a list of level-wise items, for those items which are used in the subsetting operations. If data item is used in data selection or generation at a certain level, but it is not available in the table at a lower level then the following query should be generated: "WILL THERE BE ONLY ONE COST-MASTER (or the data set used) FOR A UNIQUE PRODUCT-CODE (or set of ITEM 1, ITEM 2, ... for all the entries in the table for lower levels)?" However, if the subsetting

is due to an element level before such use, by a 'FOR EACH data-set' clause. Then no message is generated.

3. If the answer to the query, generated in the previous step is 'yes' then it becomes an integrity constraint to be checked during creation or update of COST-MASTERs.

4. If the answer is 'no' then a new query "IF THERE ARE MORE THAN ONE COST-MASTERS (or data set used) HOW TO SELECT SELLING-PRICE (or the data-item which generated this query)?" is generated.

One more problem in the example could be identified. A COST-MASTER may not be present for a PRODUCT-CODE or an invoice may have a wrong PRODUCT-CODE. In data processing such checks have to be made. User may not be aware of this though his organisation may be making a manual check for such a problem. This can be analysed in the following way.

1. Do the following for every data transformation module and for every data set. Identify all the data subsetting classes which use data selection such as COST-MASTER (PRODCT-CODE = PRODUCT-CODE OF INVOICE). For these transformations do the following 4 steps.

2. Scan further description to find out if the user has checked the presence and absence of the selected data set. If it is not so then do the following step.

3. Insert a query just after the statement. "IF THE COST-MASTER (data set) FOR PRODUCT-CODE = PRODUCT-CODE OF INVOICE (the expression) IS ABSENT? CAN IT BE ABSENT?

4. If the answer is 'No' then ask the user to insert an integrity check somewhere, if possible.

5. If the answer is 'Yes' to query in Step 3 the user will introduce the required data transformation(s), if any.

5-2. DATA CARDINALITY

The structure of the language is such that it makes the user aware of the sets and elements at any stage. The user is discouraged from thinking in terms of embedded and repeating groups. If the cardinality of these sets is known at different levels, then this could generate good design information. For example:

```
*  FOR EACH CUSTOMER-NO OF INVOICE:S BEGIN
       COUNT INVOICE:S AS INVOICES-OF-A-CUSTOMER
       FOR EACH INVOICE BEGIN
            FOR COST-MASTER (PRODUCT-CODE = PRODUCT-CODE OF INVOICE)
                                                          BEGIN
             MAKE AMOUNT-1 = STANDARD-LABOUR-PRICE
                  - - - - -
                  - - - - -
                  etc
```

Let there be 10000 invoices, 200 customers and 20 products, approximately 500 invoices per customer and 10 products ordered per customer. Reading of COST-MASTER can be reduced from 10000 to 2000 if data selection is done product-wise. Such areas could be located by the following method.

1. Select data description modules with two or more data sets.
2. Ask the user to give average volume of data in those data sets

   "APPROXIMATELY HOW MANY INVOICES AT THIS POINT"
   (data set)

3. If there is an order of difference between the two volumes then analyse as follows.

†4. Scan and locate the first data selection for a second data set. *Scan backwards from that point, by ignoring the procedures for calculations and any 'FOR EACH data set ' clause. Scan back only till some other operation is found

5. If no 'EACH data set' clause was ignored in this process then abandon the analysis for the current data selection. Otherwise query the cardinality at the point of current scan as follows:

'HOW MANY DIFFERENT VALUES HAS PRODUCT-CODE (the data item of data selection being analysed) FOR SET OF INVOICES (data set name at current scan point) AT THIS LEVEL?"

6. 'AT THIS LEVEL' part can be refined specifically giving all the selection data items of the subset class at this stage like

"INVOICES FOR A GIVEN CUSTOMER-NO?"

7. If the answer is more than one then data selection under analysis can be shifted to just below the point of back scan.

5-3. ANALYSIS OF UPDATE

The system description often has omissions about strict update sequences, deletion facility, and the cycle of update. Updates are generally used on data which is about entities with long life time, data which does not become obsolete after a time cycle and data about which deletion,update etc. are needed as a result of the environmental conditions. Sequence of different types of updates also

should be specific. For example every month the selling price may be updated for a product, if there is data for such update. Suppose, due to some policy of investment, returns and competition, it has been decided to slash the prices by one percent every six months. In this case, the precedence of six monthly and monthly updates will have to be specified. One way of specifying it by declaring it as a data transformation at the beginning of the six monthly cycle. Such a declaration will implicitly make use of the convention that the beginning of a smaller cycle is later than that of the bigger cycle. For example, monthly processing at the beginning of a month will conventionally be done before the daily processing, on the day of the beginning of the month.

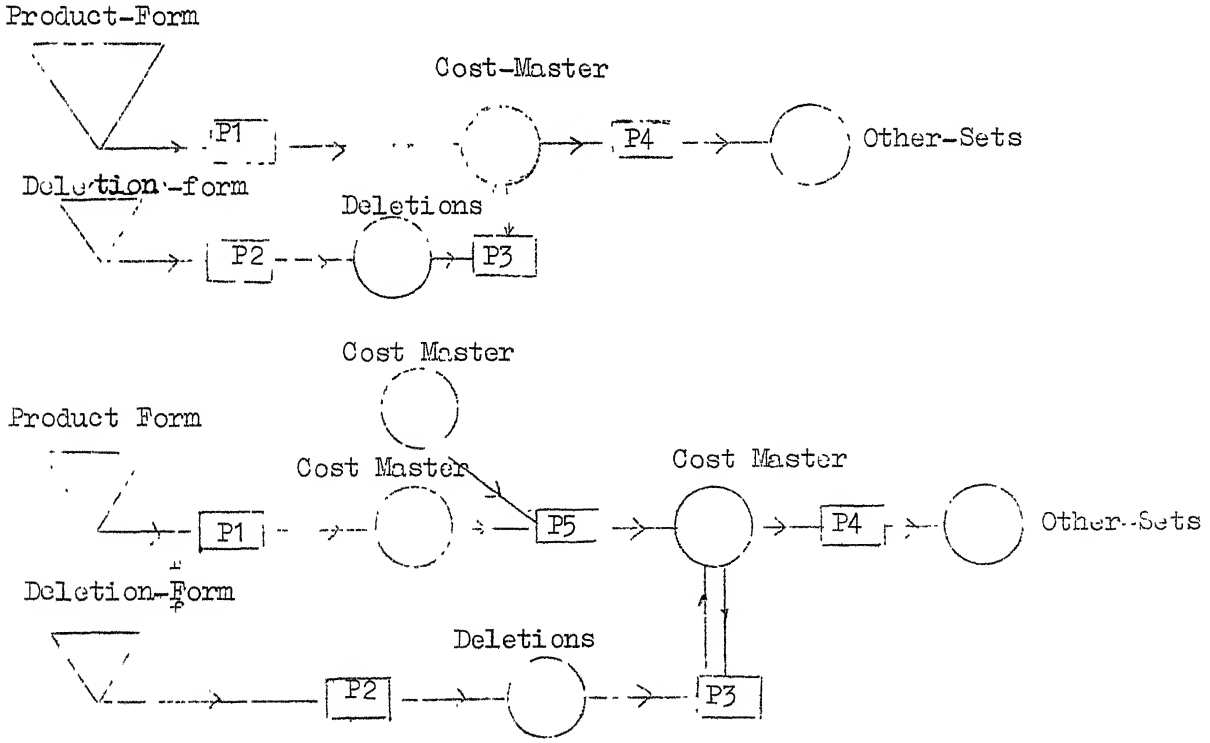


Figure 5-1: Updating Data Transformation.

Another problem in describing updates is illustrated in Figure 5-1. This problem has also been pointed at the end of the Section 5-1. The problem is that of identifying whether all the COST-MASTERS are newly created every day or only those for new products are created and added to the ones already present. This problem can be algorithmically identified from the descriptions in the following way.

1. Do the following for all data sets. Check if an UPDATE, ADD or DELETE statement is used. If these are used, then mark this data set as an entity oriented data set.
2. Out of the remaining data sets select those which are being generated directly from an input form. If the data items from any of these sets are being used over a number of time cycles then this is likely to be an entity oriented set.
3. For all the data sets marked so far as entity oriented sets do the following steps.
4. Check the cardinality handled by the creating process by creating the query:

   "APPROXIMATELY HOW MANY COST-MASTERS (data set name) ARE CREATED EVERY TIME?"

5. Check the cardinality handled by the using processes by creating the query:

   "APPROXIMATELY HOW MANY COST-MASTERS (data set name) ARE THERE?"

6. If the answer in Step 5 is greater than twice the answer in Step 4 then this data set is likely to be a growing set updated at times.
7. Take all the creating processes for the data set one by one and generate the following query "IS THIS COST-MASTER DATA TO BE ADDED, TO THE OTHER COST-MASTER(data set name)DATA THAT IS ALREADY THERE?"
8. If the answer to query in Step 7 is yes then change the CREATE to ADD and display to the user. If the user finds it unsuitable then he may give the correct version. If he accepts it then one more display may be made as in Step 9.
9. "THE VERY FIRST TIME THIS WILL BE CREATING A NEW COST-MASTER ALSO."

5-4. CONCLUSION

In conclusion it can be said that the structure of the SDL can be exploited to algorithmically analyse the system description. Such analysis has two aims. On the one hand it tries to get a logically correct and complete description and on the other hand it can gather information for design and introduce changes to make it suitable for data processing.

Some methods of such analysis have been developed in this chapter. More such analysis can be systematically done if important data processing aspects of any description can be conceptualised.

From the design point of view, it can be said that better optimization can be performed at a level at which the structure or the problem or the intent of the user is more clear. Designing of data partitions and processing programs is a difficult problem of optimization because for a data set, the organization and the accessing method of a computation using it, can not be determined independently of each other or of other data set organisations and computation accessing methods.

The organisation of a data set limits the ways in which it can be practically accessed by a computation and the accessing method of a computation restricts the practicable organisation of a data set that it accesses. But business systems are highly sequential and the problem structure can be interactively used to collect the design information from the user.

CHAPTER 6

A CASE STUDY

System description language should provide ease of use and a repertoire of concepts to describe all the relevant aspects of a number of users' systems. In this chapter a case study is developed using the system description language presented in this thesis. A case study is taken from [CLI 1]. Due to some discrepancies found in the problem analysis, the problem has been modified. The selected case study becomes difficult in terms of data descriptions, at certain points and illustrates that without the use of a host language, the system can be completely described in SDL.

6-1. PROBLEM DESCRIPTION

A complete word statement of the case study is not being given here as it will take a lot of space and also it is a need of the formal description to be readable. Certain points which are important in the case study are being described here.

In the case study given here an information system is needed for a bakery to expedite implementation of changes in the customer requirements and orders. Every day a list of all the commodities which are to be baked, is created. Bakery is used during the day to prepare confectionary items and during the night bread. Therefore, confectionary items to be sold the next day are loaded in the bakery in the morning.

Every day a list of the commodities in short supply is prepared. Orders related to these commodities are eliminated before the bakery load is prepared. After the items are ready in the bakery they are sent to six loading bays in sufficient quantity for each bay. There are delivery vans which have an associated round number (identifying the route allocated to a van) and a loading bay from which it gets loaded. Each customer is associated with a fixed round number according to the address and location.

Vans make one or two deliveries on the same route according to the ordering requirements of the day. These vans have delivery-men incharge of the van who are given a delivery note for the day to deliver the orders to the customers. The orders which are not fulfilled due to non-availablity of the commodity are notified to the customer. In case certain items are not taken by a customer, inspite of the order or are taken extra by a customer, if available, then the delivery-men make a note of it and fill the returns and adjustment forms to be submitted to the bakery, immediately.

The more complicated aspect is that of order collection which is also done by delivery-men. Orders are accepted on preprinted forms, OMR documents where the delivery man makes only a mark on the code provided. It has been done to simplify things for delivery-men

About 50 lines are accomodated on each form. There are about 120 confectionary items which take three preprinted forms and one preprinted form for bread items is sufficient.

Every week, preprinted forms are created by the computer by additionally printing the day and the week on the preprinted form. For each day, the forms are printed. These are used over the week. Each form can be used to create the orders for all 6 days of the week. The commodity is identified by the line number on the form. If the order is temporary it is a temporary change to the standing orders, otherwise a permanent change. The system maintains standing orders for customers for all six days of the week for all the required commodities. Confectionary items have to be ordered two days before whereas bread items have to be ordered just one day in advance.

6-2. INFORMATION SYSTEM DESCRIPTION

The relevant inputs, reports and data associations are described in the following pages. The documents and reports are partially described in figures to assist the reader. A flow diagram is being given in Figure 6-1 and Figure 6-2 to show the data transformations.

Figure 6-1
User's View of Overall Information Flow
(continued in Figure 6-2)

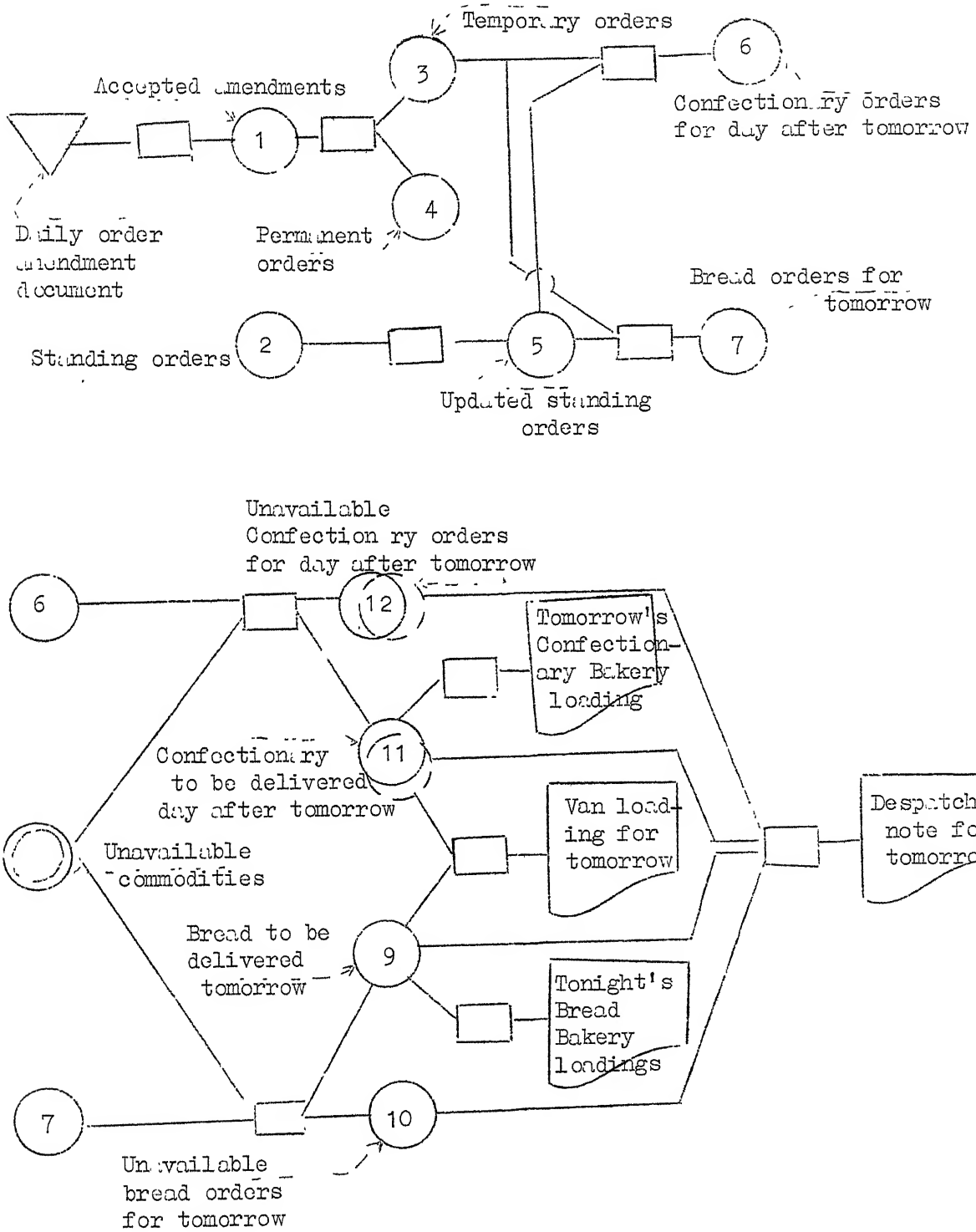
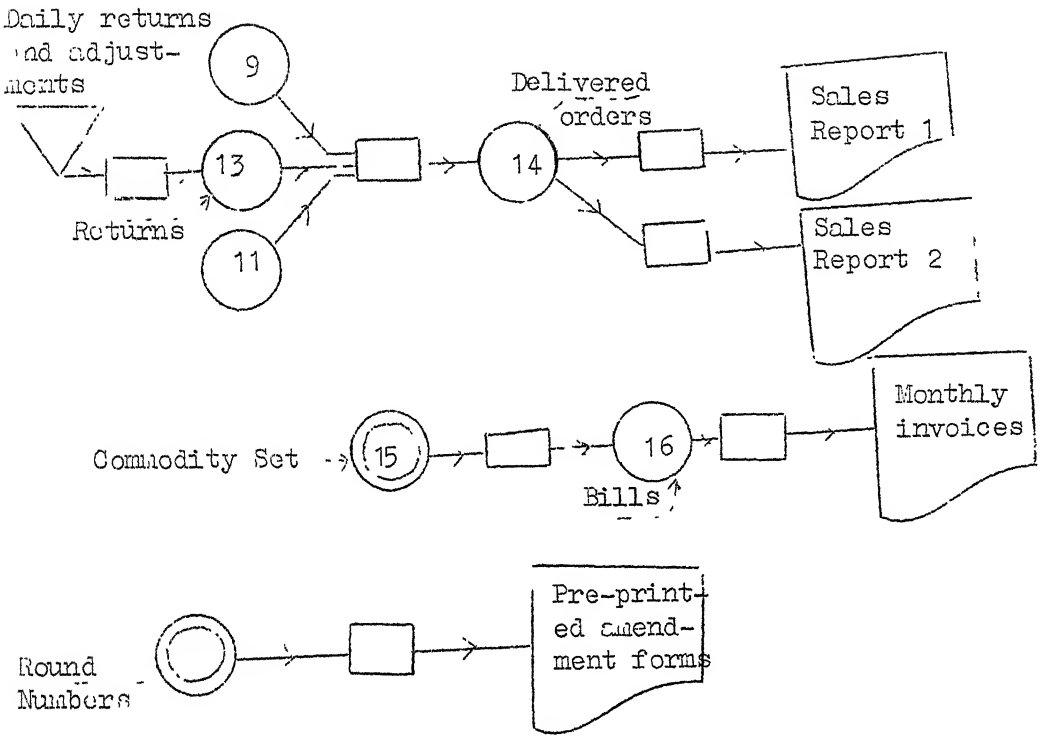


Figure 6-2
User's View of Overall Information Flow
(Continued from Figure 6-1)

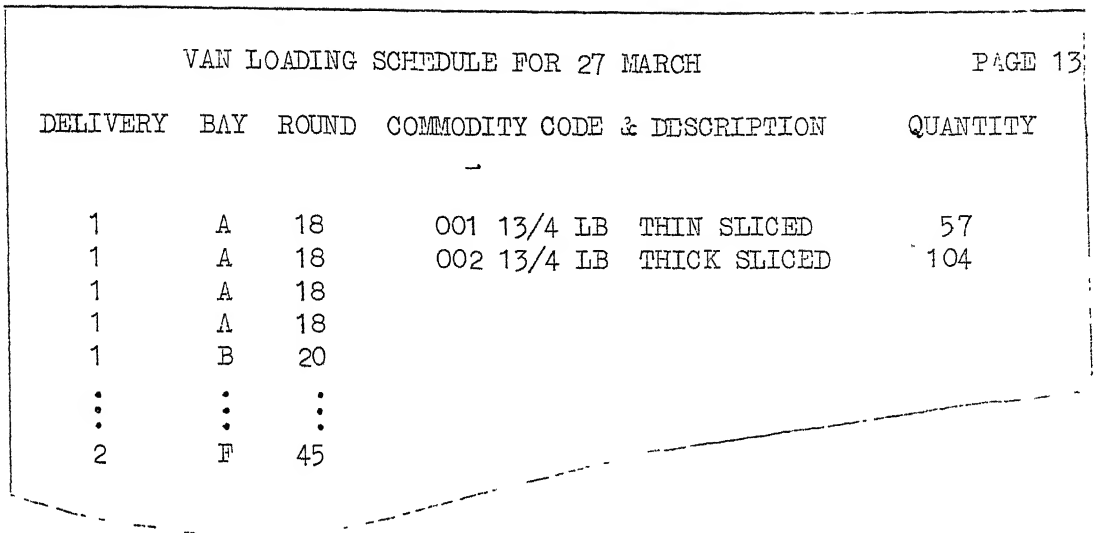



Data transformation


Data set


Data set only referenced


Versions of a data set



VAN LOADING SCHEDULE FOR 27 MARCH PAGE 13

DELIVERY	BAY	ROUND	COMMODITY CODE & DESCRIPTION	QUANTITY
1	A	18	001 13/4 LB THIN SLICED	57
1	A	18	002 13/4 LB THICK SLICED	104
1	A	18		
1	A	18		
1	B	20		
:	:	:		
2	F	45		

```
FOR EACH DAY
REPORT VAN-LOADING-SCHEDULE
FROM DATA SET DESPATCHED-ORDER:S OF TOMORROW CALLED ORDER:S
  FOR EACH ROUND OF ORDER:S, ASCENDING BEGIN
    FOR EACH DELIVERY-NO OF ORDER:S, ASCENDING BEGIN
      FOR EACH COMMODITY OF ORDER:S BEGIN
        FOR NEW-PAGE BEGIN
          VAN-LOAD-HEADING, TOMORROW (DAY OF MONTH BY NUMBER ,
          MONTH OF YEAR BY NAME), PAGE-NUMBER
          NEW-LINE, "DELIVERY BAY ROUND",
          "COMMODITY CODE & DESCRIPTION QUANTITY" END
      DELIVERY-NO, BAY-NO, ROUND, COMMODITY.
      COMMODITY-DESCRIPTION, TOTAL OF QUANTITY END
FORMAT "VAN LOADING SCHEDULE FOR" PAGE-NUMBER FROM 45,
DELIVERY FROM 4, BAY FROM 10, ROUND FROM 16, COMMODITY FROM 20,
COMMODITY-DESCRIPTION FROM 26, QUANTITY FROM 45
OVER
```

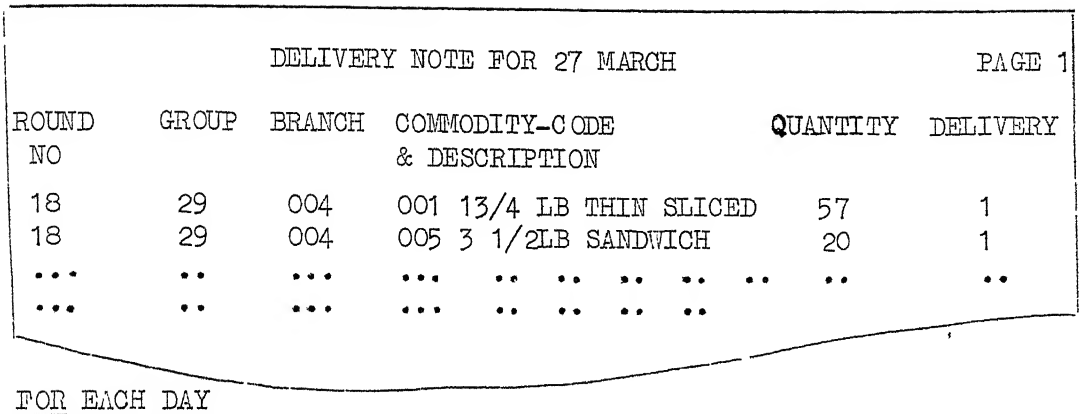
```
                BAKERY LOADING SCHEDULE FOR 27 MARCH                PAGE 1

DAY-LOADING

COMMODITY CODE & DESCRIPTION          BAY          QUANTITIES

  028      THREE  IN  ONE             A              4975
                                      B              4168
                                      C              3430
                                      D              1800
                                      E              2445
                                      F               903
                                    TOTAL           17721
  031      SNOOPY SMALL PASTRY        A              1970
                                      B                 3
                                      C
```

```
FOR EACH DAY
REPORT DAY-BAKERY-LOADING FROM DATA SET
CUSTOMER-DELIVERY-CONFECTIONARY:S OF DAY AFTER TOMORROW
  FOR EACH COMMODITY OF CONFECTIONARY-ORDER:S CALLED ORDER:S BEGIN
    FOR (1 TO 6) LINES BEFORE END-OF-PAGE BEGIN
        HEADING, TOMORROW (DAY OF MONTH BY NUMBER, MONTH BY NAME)
        PAGE-NUMBER, NEW-LINE, "DAY LOADING", NEW-LINE
        "COMMODITY CODE & DESCRIPTION     BAY     QUANTITIES" END
    NEW-LINE, COMMODITY, COMMODITY-DESCRIPTION
    FOR EACH BAY BEGIN
        BAY, TOTAL OF QUANTITY OF ORDER:S AS BAY-TOTAL, NEW-LINE END
    TOTAL-HEADING, TOTAL OF QUANTITY OF ORDER:S AS COMMODITY-TOTAL END
FORMAT HEADING FROM 10 VALUE "BAKERY LOADING SCHEDULE FOR"
PAGE-NUMBER FROM 45, COMMODITY SIZE 3, COMMODITY-DESCRIPTION
FROM 10 SIZE 20, BAY FROM 25 SIZE 1, BAY-TOTAL FROM 30
SIZE 4, TOTAL-HEADING FROM 23 SIZE 5 COMMODITY-TOTAL FROM
29 SIZE 5  OVER
```

BAKERY LOADING SCHEDULE FOR 27 MARCH PAGE 1
NIGHT-LOADING

COMMODITY	CODE & DESCRIPTION	BAY	QUANTITIES
001	1 3/4 LB THIN SLICED	A	3975
		B	1168
		C	4430
		D	2440
		E	1800
		F	304
		TOTAL	17122
002	1 3/4 LB THICK SLICED	A	1790
		B	4
		C	

FOR EACH DAY
REPORT NIGHT-BAKERY-LOADING SAME AS DAY-BAKERY-LOADING REPORT
USING (CUSTOMER-DELIVERY-BREAD:S OF TOMORROW
FOR CUSTOMER-DELIVERY-CONFECTIONARY:S OF DAY AFTER TOMORROW,
"NIGHT-LOADING" FOR "DAY-LOADING")
OVER



DELIVERY NOTE FOR 27 MARCH PAGE 1

ROUND NO	GROUP	BRANCH	COMMODITY-CODE & DESCRIPTION	QUANTITY	DELIVERY
18	29	004	001 13/4 LB THIN SLICED	57	1
18	29	004	005 3 1/2LB SANDWICH	20	1
...	..	...	... .	..	..
...	..	...	... .		

```
FOR EACH DAY
REPORT DELIVERY-NOTE
†FROM DATA SET DESPATCH-NOTE:S OF TOMORROW
FOR EACH ROUND-NO, ASCENDING BEGIN
  FOR EACH (GROUP, BRANCH, COMMODITY) ASCENDING BEGIN
    FOR NEW-PAGE BEGIN
      NOTE-HEADING, TOMORROW (DAY OF MONTH BY NUMBER, MONTH BY NAME)
      PAGE-NUMBER, NEW-LINE, PAGE-HEADING, NEW-LINE, HEADING-2 END
   *ROUND-NO, GROUP, BRANCH
    COMMODITY, COMMODITY-DESCRIPTION, QUANTITY, DELIVERY END
FORMAT NOTE-HEADING VALUE "DELIVERY NOTE FOR" FROM 15,
PAGE-HEADING VALUE "ROUND-NO GROUND BRANCH COMMODITY CODE"-
-"   QUANTITY DELIVERY", HEADING-2 "& DESCRIPTION"
FROM 46, ROUND-NO SIZE 2 FROM 2, GROUP FROM 9 SIZE 2 BRANCH FROM 13SIZE 3,
COMMODITY FROM 19 SIZE 3 COMMODITY-DESCRIPTION FROM 23 SIZE 20 QUANTITY
FROM 43 SIZE 2, DELIVERY FROM 50 SIZE 1 OVER
```

† Data selection is based on day as a time cycle.

* This report can accomodate only one dispatch note for a day, for a given ROUND-NO, GROUP, BRANCH and COMMODITY and it can be an integrity constraint of the system.

HARTEE BAKERIES LTD

ROUND NO.	WEEK/DAY	ROUND-No	WEEK NO	DAY NO.	CODE
27	23 FRI	- - - -	- -	- -	

(PREPRINTED FORMS)

```
FOR EACH WEEK
REPORT ORDER-AMENDMENTS-FOR-BREAD FROM DATA SET ROUND-NUMBER
FOR EACH ROUND-NO ASCENDING BEGIN
  FOR EACH DAY OF WEEK BEGIN
    FOR 50 TIMES BEGIN
      NEW-PAGE AFTER 5 LINES
      ROUND-NO, WEEK OF YEAR BY NUMBER, DAY BY NAME,
     *ROUND-CODE, WEEK-CODE, DAY-CODE, ITEM-CLASS  END
FORMAT ROUND-NO FROM 6, WEEK FROM 11, DAY FROM 14, ROUND-CODE FROM 18
SIZE 6, WEEK-CODE FROM 24 SIZE 6 DAY-CODE FROM 30 SIZE 3
ITEM-CLASS VALUE ZERO SIZE 2 FROM 34 OVER

FOR EACH WEEK
REPORT ORDER-AMENDEMENTS-FOR-CONFECTIONARY-1 SAME AS
ORDER-AMENDMENTS-FOR-BREAD REPORT USING (" -" FOR ITEM-CLASS) OVER

FOR EACH WEEK
REPORT ORDER-AMENDMENTS-FOR-CONFECTIONARY-2 SAME AS
ORDER-AMENDMENTS-FOR-BREAD REPORT USING ("- " FOR ITEM-CLASS) OVER

FOR EACH WEEK REPORT ORDER-AMENDMENTS-FOR-CONFECTIONARY-3
SAME AS ORDER-AMENDMENTS-FOR-BREAD REPORT USING ("-- " FOR ITEM-CLASS) OVER
```

*Encoding data transformations are needed for this.

SALES REPORT FOR AGGREGATE COMMODITY RANGE PAGE 1
WITH INDIVIDUAL CUSTOMER GROUPS
AND INDIVIDUAL WEEKLY VALUES

COMMODITY CODE	CUSTOMER GROUP	WEEK 5	WEEK 6	WEEK 7	WEEK 8
100-125	10	11603	10243	9178	8413
	12	6451	5218	4421	4845
	15	782	660	1199	993
	16	1395	921	996	
	22	536	680	770	
	25	129	230		
	26	488			

```
FOR EACH MONTH
REPORT SALES-OVER-WEEKS FROM DATA SET DELIVERED-ORDERED:S OF WEEK-SET-1*
  SUB-REPORT-1 REPORT
  FOR EACH COMMODITY-SET BEGIN
    NEW-LINE, SPACE 2, COMMODITY-CODES (SIZE 7)
    FOR EACH GROUP ASCENDING BEGIN
      GROUP
      FOR EACH WEEK-OF-DELIVERED-ORDER:S,ASCENDING (4 COLUMNS OF
 ††   SIZE 16 FROM 36) BEGIN TOTAL OF QUANTITY END
      NEW-LINE
      FOR NEW-PAGE BEGIN
        SUB-REPORT-1 REPORT
        NEW-LINE, †COMMODITY-SET END
FORMAT COMMODITY-SET FROM 2 SIZE 7, GROUP FROM 14 SIZE 2 TOTAL FROM
10 SIZE 6  OVER
```

*WEEK-SET-1 is a user invented time duration, to be described.

†If group number continue on a new page then COMMODITY-SET is to be repeated once more.

††Multiple instances of data may be formatted vertically or horizontally. For horizontal formatting a page may be divided into repeating columns.

```
**REPORT SUB-REPORT-1 FROM DATA SET NIL
  FOR NEW-PAGE BEGIN
    BLANK 2, "SALES REPORT FOR AGGREGATE COMMODITY RANGE"
    SPACE 10, PAGE-NUMBER, NEW-LINE, SPACE 14,
    "WITH INDIVIDUAL CUSTOMER GROUPS", NEW-LINE, SPACE 16,
    "AND INDIVIDUAL WEEKLY VALUES" NEW-LINE, SPACE 2,
    "COMMODITY CUSTOMER WEEK5 WEEK6 WEEK7 WEEK8"
    NEW-LINE, SPACE 4 "CODES GROUP" END
  OVER
```

** New page description being a lengthy description, is difficult to repeat and is described as a secondary report which does not have an associated condition for its production.

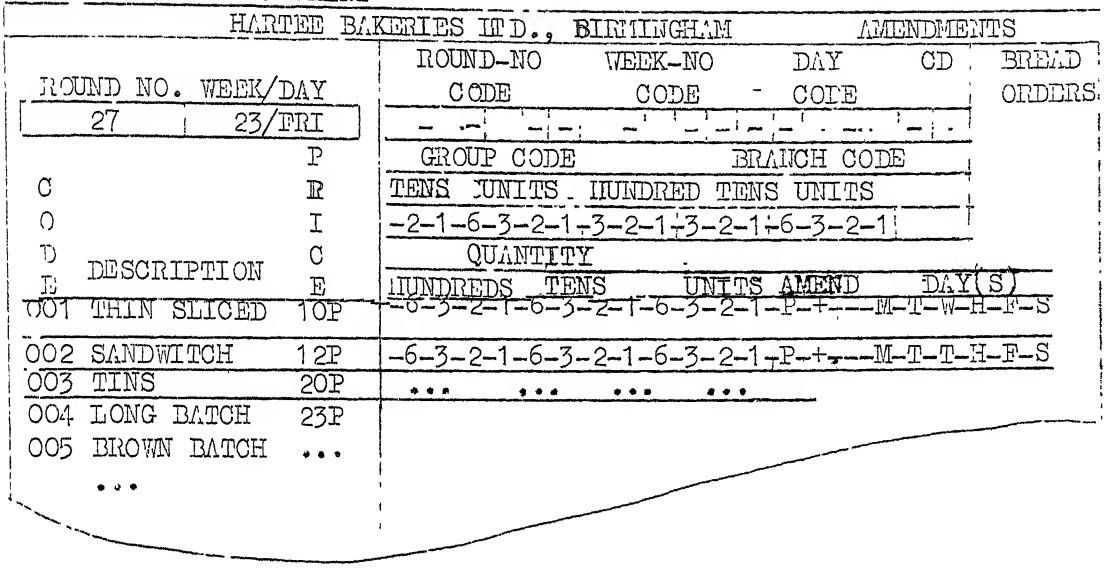
```
                SALES REPORT FOR SEPARATE COMMODITIES                PAGE 1
                  WITH INDIVIDUAL CUSTOMER GROUPS AND
                       AND AVERAGE QUANTITIES

COMMODITY        CUSTOMER          AVERAGE OVER
  CODE             GROUP           MONTH  13-24

   104              00               156204
   104              01                30821
   104              02                14528
   108              00               116983
   108              01                1240-
```

```
FOR EACH MONTH
REPORT COMMODITY-SALES FROM DATA SET DELIVERED-ORDER:S OF MONTH-SET-1
FOR EACH COMMODITY, GROUP-NO, ASCENDING BEGIN
  FOR NEW-PAGE BEGIN
    HEADING-1, PAGE-NUMBER, NEW-LINE,HEADING-2
    NEW-LINE,HEADING-3 END
  COMMODITY
  GROUP-NO
  TOTAL OF QUANTITY END
FORMAT
HEADING-1 VALUE "SALES REPORT FOR SEPARATE COMMODITIES" FORM 8
HEADING-2 FROM 10 VALUE "WITH INDIVIDUAL CUSTOMER GROUPS"
HEADING-3 VALUE "AND AVERAGE QUANTITIES" FROM 15 COMMODITY SIZE 3 FROM 9
GROUP-NO SIZE 2 FROM 19 TOTAL SIZE 10 FROM 24
OVER
```

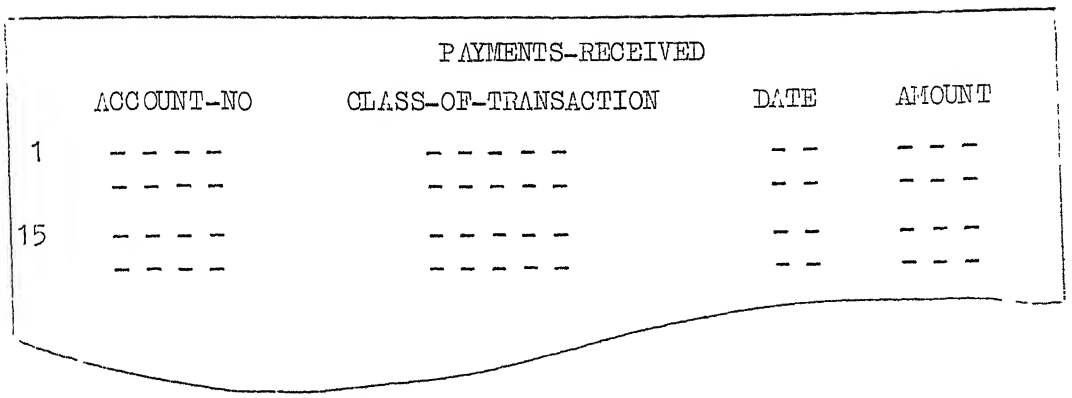
```
                              MONTHLY INVOICE

INVOICE NO:  3935     BILLS: 35                                      MARCH
CUSTOMER NAME : A.D. MICHAEL
CUSTOMER ADDRESS : FLAT 29, STREET 50,
                   LONDON
S.N.  DATE          COMMODITY              ORDER          AMOUNT

  1   5/3/77        1 3/4LB THIN SLICED    5              21.00
```

```
FOR EACH MONTH
REPORT INVOICES FROM DATA SET BILLS OF CURRENT MONTH
FOR EACH CUSTOMER-CODE BEGIN
  NUMBER ON DATE ASCENDING AS BILL-NUMBER, CURRENT MONTH BY LINE
  INCREMENT INVOICE-NO BY 1
  COUNT BILLS AS BILL-COUNT
  NEW-PAGE, SPACE 10, "MONTHLY INVOICE" NEW-LINE
  "INVOICE NO : " INVOICE-NO," BILLS:  " BILL-COUNT
  NEW-LINE, "CUSTOMER NAME :  " CUSTOMER-NAME
  NEW-LINE, "CUSTOMER ADDRESS  : " ADDRESS-LINE-1,
  NEW-LINE, ADDRESS-LINE-2, NEW-LINE, HEADING
  FOR EACH BILL BEGIN
    FOR NEW-PAGE BEGIN
      "INVOICE NO:   " INVOICE-NO
      NEW-LINE, HEADING END
    NEW-LINE, BILL-NUMBER, DATE, COMMODITY
    QUANTITY, VALUE END
  NEW-LINE, "INVOICE NO" INVOICE-NO,
  SPACE 3, "TOTAL" TOTAL OF AMOUNT
  FORMAT INVOICE-NO SIZE 4, BILL-COUNT SIZE 2,
  CUSTOMER-NAME SIZE 20, ADDRESS-LINE-1 SIZE 30
  ADDRESS-LINE-2 SIZE 30, HEADING
  "S.N.     DATE          COMMODITY       ORDER          AMOUNT "
  BILL-NUMBER SIZE 2, DATE FROM 4 SIZE 8, COMMODITY
  FROM 14 SIZE 25 QUANTITY FROM 40 SIZE 2, VALUE
  FROM 44 PICTURE 99.99, TOTAL PICTURE 9999.99 OVER
```

PREPRINTED OMR DOCUMENT




```
FOR EACH DAY
DOCUMENT ORDER-AMENDMENTS TO DATA SET TYPE-B1:S
FOR EACH 50 TYPE-B1:S BEGIN REQUIRED ALL
   NEW-FORM, AFTER 5 LINES
   ROUND-NO, WEEK-NO, DAY-NO, ROUND-NO-CODE, WEEK-NO-CODE
   DAY-NO-CODE, ITEM-CLASS, NEW-LINE, GROUP-CODE, BRANCH-CODE
   FOR EACH OF TYPE-B1 : BEGIN
      NEW-LINE
      FOR B-1-DATA = SPACE BEGIN UNFILLED, IGNORE END
      FOR  B-1-DATA NOT = SPACE BEGIN
         REQUIRED ALL, NEW-LINE
         MAKE COMMODITY-LINE = LINE-NUMBER,
         QUANTITY-CODE, AMENDMENT-CODE, AMENDMENT-TYPE,
         AMENDMENT-DAYS END
FORMAT ROUND-NO FROM 5 SIZE 2, WEEK-NO FROM 10 SIZE 2 VALUE CURRENT
WEEK OF YEAR BY NUMBER, DAY-NO FROM 12 SIZE 3 VALUE CURRENT
DAY BY NAME ROUND-NO-CODE FROM 17 SIZE 6 WEEK-NO-CODE FROM
23 SIZE 6, DAY-NO-CODE FROM 29 SIZE 3-ITEM-CLASS FROM 32 SIZE 2
GROUP-CODE FROM 17 SIZE 6 BRANCH-CODE FROM 23 SIZE 6
QUANTITY-CODE FROM 17 SIZE 12 AMENDMENT-CODE FROM 29
SIZE 1, AMENDMENT-TYPE FROM 30 SIZE 2 AMENDMENT-DAYS FROM 32
SIZE 6, B-1-DATA FROM 17 SIZE 21
OVER
```




```
FOR EACH MONTH
DOCUMENT PAYMENT-RECEIVED TO DATA SET TYPE-B5
FOR EACH 15 TYPE-B5:S BEGIN
          NEW-FORM, AFTER 3 LINES
          FOR EACH TYPE-B5 BEGIN
               NEW-LINE
               FOR B5-DATA = BLANK BEGIN UNFILLED, IGNORED END
           FOR B5-DATA NOT = BLANK BEGIN
               REQUIRED ALL, ACCOUNT-NO, CLASS-OF-TRANSACTION
               DATED, AMOUNT END
FORMAT B5-DATA FROM 2 SIZE 26, ACCOUNT-NO FROM 2 SIZE 6 VALUE NUMBER,
CLASS-OF-TRANSACTION FROM 9 SIZE 1 VALUE RANGE 1 TO 6, DATED FROM 11
SIZE 6 VALUE (ZERO TO CURRENT DATE) AMOUNT FROM 18 PIC 999.99 VALUE
NUMBER OVER

FOR EACH DAY
DOCUMENT RETURN-ADJUSTMENTS SAME AS ORDER-AMENDMENTS
DOCUMENT USING (NIL FOR AMENDMENT-CODE, TYPE-3 FOR TYPE-1)
OVER
```

```
DATA-DESCRIPTION

BREAD-ORDER:S  DEFINED AS ORDER:S
CONFECTIONARY-ORDER: DEFINED AS ORDER:S
OVER


DATA-DESCRIPTION
PERIOD WEEK-SET-1 IS FROM 8 WEEKS BEFORE CURRENT-WEEK
                     TO   4 WEEKS BEFORE CURRENT-WEEK
                     OVER


DATA-DESCRIPTION
FOR ORDER:S BEGIN
  FOR EACH ORDER BEGIN
    FOR COMMODITY OF ORDER ≥ 100 OR ≤ 125 BEGIN
      MAKE COMMODITY-SET - "100-125" END
    FOR COMMODITY OF ORDER >125 BEGIN
      MAKE COMMODITY-SET = "125-165" END
OVER


DATA-DESCRIPTION
PERIOD MONTH-SET-1 IS FROM 24 MONTHS BEFORE CURRENT-MONTH
                      TO 12 MONTHS BEFORE CURRENT MONTH
                      OVER
```

COMMENT: There is very complicated decoding of ORDER form and therefore the next few data description modules happen to be mostly procedures.

```
DATA-DESCRIPTION DECODING-ORDERS FROM DATA SET TYPE-B1 OF TODAY
FOR EACH TYPE-B1 :S BEGIN
    SAME AS BINARY DATA-DESCRIPTION USING (ROUND-NO-CODE FOR CODE,
    ROUND-NO FOR VALUE)
    SAME AS BINARY DATA-DESCRIPTION USING (WEEK-NO-CODE FOR CODE,
    WEEK-NO FOR VALUE)
    SAME AS BINARY DATA-DESCRIPTION USING (DAY-NO-CODE
    FOR CODE, DAY-NO FOR VALUE)
    SAME AS BINARY DATA-DESCRIPTION USING (ITEM-CLASS
    FOR CODE, ITEM-NO FOR VALUE)
    MAKE GROUP-NO = 0
    SAME AS CODE-21 DATA-DESCRIPTION USING (POSITION 1 TO 2 OF GROUP-CODE
    FOR CODE, GROUP-NO FOR VALUE)
    SAME AS CODE-6321 DATA-DESCRIPTION USING (POSITION 3 TO 6
    OF GROUP-CODE FOR CODE, GROUP-NO FOR VALUE)
    MAKE BRANCH-NO = 0
    SAME AS CODE-321 DATA-DESCRIPTION USING (POSITION 1 TO 3
    OF BRANCH-CODE FOR CODE, BRANCH-NO FOR VALUE)
    SAME AS CODE-321 DATA-DESCRIPTION USING (POSITION 4 TO 6
    OF BRANCH-CODE FOR CODE, BRANCH-NO FOR VALUE)
    SAME AS CODE-6321 DATA-DESCRIPTION USING (POSITION 7 TO 10 OF
    BRANCH-CODE FOR CODE, BRANCH-NO FOR VALUE)
    MAKE QUANTITY = 0
    SAME AS CODE-6321 DATA-DESCRIPTION USING (POSITION 1 TO 4 OF
    QUANTITY-CODE FOR CODE, QUANTITY FOR VALUE)
    SAME AS CODE-6321 DATA-DESCRIPTION USING (POSITION 5 TO 8 OF
    QUANTITY-CODE FOR CODE, QUANTITY FOR VALUE)
    SAME AS CODE-6321 DATA-DESCRIPTION USING (POSITION 9 TO 12 OF
    QUANTITY-CODE FOR CODE, QUANTITY FOR VALUE)
    CREATE DECODED-B1 (ROUND-NO, WEEK-NO, DAY-NO, ITEM-NO, GROUP-NO,
    BRANCH-NO, QUANTITY)
    END OVER
```

COMMENT: Reference to a part of a data item can be made by POSITION. The same data could have been described in the input document by using a number of data names, to avoid a partial reference.

```
DATA-DESCRIPTION  BINARY
MAKE VALUE = 0
FOR POSITION 1 of CODE = 1 BEGIN ADD 32 TO VALUE END
FOR POSITION 2 OF CODE = 1 BEGIN ADD 16 TO VALUE END
FOR POSITION 3 OF CODE = 1 BEGIN ADD  8 TO VALUE END
FOR POSITION 4 OF CODE = 1 BEGIN ADD  4 TO VALUE END
FOR POSITION 5 OF CODE = 1 BEGIN ADD  2 TO VALUE END
FOR POSITION 6 OF CODE = 1 BEGIN ADD  1 TO VALUE END
OVER

DATA-DESCRIPTION CODE-6321
MAKE CHECK = 0
FOR POSITION 1 OF CODE = 1 BEGIN ADD  6 TO CHECK END
FOR POSITION 2 OF CODE = 1 BEGIN ADD  3 TO CHECK END
FOR POSITION 3 OF CODE = 1 BEGIN ADD  2 TO CHECK END
FOR POSITION 4 OF CODE = 1 BEGIN ADD  1 TO CHECK END
ADD CHECK TO VALUE  OVER

DATA-DESCRIPTION GET-VALUE
FOR CHECK < 10 BEGIN MULTIPLY CHECK BY 10
                     ADD  CHECK TO VALUE END
OVER

DATA-DESCRIPTION CODE-321
MAKE CHECK = 0
FOR POSITION 1 OF CODE = 1 BEGIN ADD 3 TO CHECK END
FOR POSITION 2 OF CODE = 1 BEGIN ADD 2 TO CHECK END
FOR POSITION 3 OF CODE = 1 BEGIN ADD 1 TO CHECK END
SAME AS GET-VALUE DATA-DESCRIPTION
OVER

DATA-DESCRIPTION CODE-21
MAKE CHECK = 0
FOR POSITION 1 OF CODE = 1 BEGIN ADD 2 TO CHECK END
FOR POSITION 2 OF CODE = 2 BEGIN ADD 1 TO CHECK END
SAME AS GET-VALUE DATA-DESCRIPTION
OVER

DATA-DESCRIPTION CODE-63211
SAME AS CODE-6321 DATA-DESCRIPTION USING (NIL FOR ADD CHECK TO VALUE)
SAME AS GET-VALUE DATA-DESCRIPTION
```

```
DATA-DESCRIPTION DECODING-THE-DAY-AND-COMMODITY-CODE
FOR DECODED-B1:S, COMMODITY-SET:S BEGIN
FOR EACH DECODED-B1 BEGIN
  FOR AMENDEMENT-TYPE ≠ "11" BEGIN
    FOR DAY-NO = CURRENT DAY OF WEEK BY NUMBER BEGIN
      FOR WEEK-NO = CURRENT WEEK OF YEAR BY NUMBER BEGIN
        FOR COMMODITY-SET (ITEM CLASS = ITEM-CLASS OF DECODED-B1,
            COMMODITY-LINE = COMMODITY-LINE OF DECODED-B1) BEGIN
            COMMODITY = COMMODITY OF COMMODITY-SET
            REDESCRIBE AMENDMENT-DAYS BEGIN
              AMENDMENT-DAY REPEATED 6 TIMES END
            NUMBER _ DECODED-B1 _ _ _ AS DAY-COUNT
            FOR EACH DECODED-B1 BEGIN FOR AMENDMENT-DAY ≠ 0 BEGIN
              CREATE ACCEPTED-AMENDMENT (DAY-COUNT)
              END  OVER
```

COMMENT: Besides the DAY-COUNT the other data items associated with ACCEPTED-AMENDMENT are being assumed by the user, for the time being.

```
DATA-DESCRIPTION NIL
FOR ACCEPTED-AMENDMENT:S OF CURRENT DAY BEGIN
FOR EACH ACCEPTED-AMENDMENT BEGIN
  FOR AMENDMENT-CODE = "1" BEGIN CREATE PERMANENT-AMENDMENT END
  FOR AMENDMENT-CODE = 0 BEGIN CREATE TEMPORARY-AMENDMENT END
OVER
```

COMMENT: The ACCEPTED-AMENDMENT at this point has two time tags. It is today's amendment because it's been accepted today. Also, it may be an amendment for a Friday order. This second time tag is kept as a data item DAY-COUNT in the description.

*Repeating of the field creates a set of 6 DECODE D-B1, for other processing statements at this level.

```
DATA-DESCRIPTION UPDATING-STANDING-ORDERS
FOR PERMANENT-AMENDMENTS, STANDING-ORDER:S BEGIN
  FOR EACH PERMANENT-AMENDMENT BEGIN
    FOR  STANDING-ORDER (GROUP-CODE, BRANCH-CODE, COMMODITY;
      DAY-COUNT = ALL OF PERMANENT-AMENDMENT) BEGIN
      FOR STANDING-ORDER PRESENT BEGIN
        FOR AMENDMENT-CODE = "10" BEGIN
          MAKE QUANTITY = QUANTITY OF STANDING-ORDER
          ADD QUANTITY OF PERMANENT-AMENDMENT TO QUANTITY END
          FOR AMENDMENT-CODE = "01" BEGIN
            FOR QUANTITY OF PERMANENT-AMENDMENT
                QUANTITY OF STANDING-ORDER BEGIN
              MAKE QUANTITY = QUANTITY OF STANDING-ORDER
              SUBTRACT QUANTITY OF PERMANENT-AMENDMENT FROM QUANTITY
              END END
          FOR AMENDMENT-CODE = "00" BEGIN
            MAKE QUANTITY = QUANTITY OF PERMANENT-AMENDMENT END
         *UPDATE STANDING-ORDER (QUANTITY = QUANTITY)
          END
        FOR STANDING-ORDER ABSENT BEGIN
          FOR AMENDMENT-CODE = "00" BEGIN
            CREATE STANDING-ORDER (QUANTITY, GROUP-CODE, BRANCH-CODE,
            COMMODITY, DAY-COUNT = ALL OF PERMANENT-AMENDMENT) END
OVER
```

COMMENT: The presence and absence of standing orders is checked here because the actions are specific. The user may forget to describe what is to be done in case of absence generally.

* QUANTITY at the right hand side may be of PERMANENT-AMENDMENTS or of STANDING-ORDER.

```
DATA-DESCRIPTION CREATING-TEMPORARY-ORDERS SAME AS UPDATING STANDING-ORDERS
DATA-DESCRIPTION USING (TEMPORARY-AMENDMENT
FOR PERMANENT-AMENDMENT,
CREATE TEMPORARY-ORDER FOR UPDATE STANDING-ORDER,
CREATE TEMPORARY ORDER FOR CREATE STANDING-ORDER)
OVER

DATA-DESCRIPTION TEMPORARY-ORDER-HANDLING
PERIOD DAY OF WEEK BY NUMBER IS DAY-COUNT,
       OF TEMPORARY-ORDER
PERIOD WEEK OF YEAR IS CURRENT WEEK, OF TEMPORARY-ORDER
PERIOD YEAR IS CURRENT YEAR, OF TEMPORARY-ORDER
PERIOD DAY OF WEEK BY NUMBER IS DAY-COUNT, OF STANDING-ORDER
OVER
```

COMMENT: The time tag declaration here specifies two things -

1. It converts a data item DAY-COUNT to time tag
2. On standing orders the time tag is only within a weekly cycle and shows that Monday's orders become today's orders every Monday. This is not so for temporary orders which exist only one Monday.

```
DATA-DESCRIPTION KINDS-OF-STANDING-ORDERS
FOR STANDING-ORDER:S BEGIN
  FOR COMMODITY < 10 BEGIN
    STANDING-ORDER:S DEFINED AS BREAD-STANDING-ORDER:S END
  FOR COMMODITY >10 BEGIN
    STANDING-ORDER:S DEFINED AS CONFECTIONARY-STANDING-ORDER:S END
OVER

DATA-DESCRIPTION KINDS-OF-TEMPORARY-ORDERS
FOR TEMPORARY-ORDER:S BEGIN
  FOR COMMODITY < 10 BEGIN
    TEMPORARY-ORDERS DEFINED AS BREAD-TEMPORARY-ORDER:S END
  FOR COMMODITY > 10 BEGIN
     TEMPORARY-ORDER:S DEFINED AS CONFECTIONARY-TEMPORARY-ORDER:S
  END
OVER

DATA-DESCRIPTION PREPARING-ORDERS-FOR-BREAD
FOR EACH DAY
FOR BREAD-TEMPORARY-ORDER:S OF TOMORROW,
    BREAD-STANDING-ORDER:S OF TOMORROW BEGIN
  FOR EACH STANDING-ORDER BEGIN
    FOR BREAD-TEMPORARY-ORDER (GROUP-CODE, BRANCH-CODE,
        COMMODITY = ALL OF BREAD-STANDING-ORDER) BEGIN
      FOR BREAD-TEMPORARY-ORDER ABSENT BEGIN
        CREATE BREAD-ORDER (ALL OF BREAD-STANDING-ORDER)
        PERIOD OF BREAD-ORDER IS TOMORROW
        END END END
  FOR EACH BREAD-TEMPORARY-ORDER BEGIN
      CREATE BREAD-ORDER (ALL OF BREAD-TEMPORARY-ORDER)
      PERIOD OF BREAD-ORDER IS TOMORROW
OVER

DATA-DESCRIPTION PREPARING-ORDERS-FOR-CONFECTIONARY
SAME AS PREPARING-ORDERS-FOR-BREAD DATA-DESCRIPTION USING
(DAY AFTER TOMORROW FOR TOMORROW, CONFECTIONARY-ORDER
 FOR BREAD-ORDER, CONFECTIONARY-TEMPORARY-ORDER FOR
 BREAD-TEMPORARY-ORDER, CONFECTIONARY-STANDING-ORDER
 FOR BREAD-STANDING-ORDER) OVER
```

```
DATA-DESCRIPTION BREAD-ORDERS-TO-BE-DELIVERED
FOR BREAD-ORDER:S OF TOMORROW BEGIN
  FOR UNAVAILABLE-COMMODITY:S OF TODAY BEGIN
    FOR EACH BREAD-ORDER BEGIN
      FOR UNAVAILABLE-COMMODITY (COMMODITY = COMMODITY-CODE
        OF BREAD-ORDER) BEGIN
        FOR UNAVAILABLE-COMMODITY PRESENT BEGIN
          CREATE UNAVAILABLE-BREAD-ORDER (ALL OF BREAD-ORDER)
          PERIOD OF UNAVAILABLE-BREAD-ORDER IS TOMORROW END
        FOR UNAVAILABLE-COMMODITY ABSENT BEGIN
          CREATE CUSTOMER-DELIVERY-BREAD = (ALL OF BREAD-ORDER)
        PERIOD OF CUSTOMER-DELIVERY-BREAD IS TOMORROW END
OVER
```

COMMENT: At this point unavailable commodities remain as a data for which no input is defined. Very low volume data often has no standard input forms.

```
DATA-DESCRIPTION CONFECTIONARY-ORDERS-TO-BE-DELIVERED
SAME AS BREAD-ORDERS-TO-BE-DELIVERED DATA DESCRIPTION USING
(CONFECTIONARY-ORDER:S FOR BREAD-ORDER:S, DAY AFTER TOMORROW FOR TOMORROW,
 CUSTOMER-DELIVERY-CONFECTIONARY FOR CUSTOMER-DELIVERY-BREAD)
OVER

DATA-DESCRIPTION ROUND-DELIVERY-NOS
FOR EACH DAY
FOR CUSTOMER-DELIVERY-BREAD:S OF TOMORROW CALLED ORDER:S,
    CUSTOMER-DELIVERY-CONFECTIONARY OF TOMORROW CALLED ORDER:S BEGIN
  FOR EACH ROUND-NO OF ORDER:S, BEGIN
    MAKE DELIVERY-NO = 1
    MAKE LOADING-SPACE = VAN-CAPACITY
    FOR EACH ORDER BEGIN
      FOR SIZE-POINTS OF ORDER > LOADING-SPACE BEGIN
          INCREMENT DELIVERY-NO BY 1
          MAKE LOADING-SPACE = VAN-CAPACITY END
      DECREASE LOADING-SPACE BY SIZE-POINT OF ORDER
      CREATE DESPATCHED-ORDER (DELIVERY = DELIVERY-NO, ALL OF ORDERS)
      PERIOD OF DESPATCH-ORDER IS TOMORROW,
      END
OVER

DATA-DESCRIPTION NIL
UNAVAILABLE-BREAD-ORDER:S DEFINED AS DESPATCH-NOTE
UNAVAILABLE-CONFECTIONARY-ORDER:S DEFINED AS DESPATCH-NOTE
DESPATCHED-ORDER DEFINED AS DESPATCH-NOTE
OVER
```

```
DATA-DESCRIPTION RETURNS-CHECKING
SAME AS DECODING-ORDERS DATA-DESCRPTION USING
(TYPE-B3 FOR TYPE-B1, DECODED-B3 FOR DECODED-B1)
OVER

DATA-DESCRIPTION DECODING-DAY-FOR-RETURNS
SAME AS DECODING-DAY DATA-DESCRPTION USING
(DECODED-B3 FOR DECODED-B1, ACCEPTED-RETURN FOR ACCEPTED-AMENDMENT)
OVER

DATA-DESCRIPTION PERIOD-OF-RETURN
FOR ACCEPTED-RETURNS OF TODAY BEGIN
  FOR DAY-COUNT OF ACCEPTED-ORDER = YESTERDAY OF WEEK BY NUMBER BEGIN
     ACCEPTED-RETURN DEFINED AS RETURN END
OVER
```

COMMENT: Here day count is not used to declare a period but for data selection. Period of RETURN by default is that of ACCEPTED-RETURN. It implies that returns have to come within a day.

```
DATA-DESCRIPTION UPDATING-DELIVERY
FOR RETURNS FOR TODAY BEGIN
  FOR DESPATCHED-ORDERS OF YESTERDAY BEGIN
    FOR EACH RETURN BEGIN
      FOR DESPATCHED-ORDER (GROUP-NO, BRANCH, COMMODITY = ALL OF RETURN)
                                                              BEGIN
        FOR DESPATCHED-ORDER PRESENT BEGIN
          FOR AMENDMENT-CODE = "10" BEGIN
            ADD QUANTITY OF RETURN TO QUANTITY OF DESPATCHED-ORDER
            GIVING NEW-QUANTITY END
          FOR AMENDMENT-CODE = "01" BEGIN
            FOR QUANTITY OF DESPATCHED-ORDER NOT < QUANTITY OF
                RETURN BEGIN
              SUBTRACT QUANTITY OF RETURN FROM QUANTITY OF DESPATCHED-ORDER
              GIVING NEW-QUANTITY END
            END
          CREATE DELIVERED-ORDER (QUANTITY = NEW-QUANTITY, ALL OF
                                  DESPATCHED-ORDER)
          PERIOD OF DELIVERED-ORDER IS YESTERDAY END
OVER
```

COMMENT: In the above description, from today's returns, yesterday's corrected DELIVERED-ORDERS have been created.

```
DATA-DESCRIPTION FINAL-DELIVERED-ORDERS
FOR DESPATCHED-ORDER:S OF YESTERDAY,
    ` DELIVERED-ORDER:S OF YESTERDAY BEGIN
    FOR EACH DESPATCHED-ORDER BEGIN
      FOR DELIVERED-ORDER:S (GROUP, BRANCH, COMMODITY
      = ALL OF DESPATCH-ORDER) BEGIN
      FOR DELIVERED-ORDER ABSENT BEGIN
      CREATE DELIVERED-ORDER = (ALL OF DESPATCHED-ORDER)
      END
OVER


FOR EACH MONTH
DATA-DESCRIPTION CREATION-OF-BILL
FOR DELIVERED-ORDER:S OF CURRENT MONTH,
      COMMODITY-SET:S BEGIN
    FOR EACH COMMODITY-SET BEGIN
      FOR DELIVERED-ORDERS (COMMODITY = COMMODITY-CODE OF COMMODITY-SET)
                                                          BEGIN
        FOR EACH DELIVERED-ORDER BEGIN
            MULTIPLY QUANTITY OF ORDER BY PRICE OF
            COMMODITY-SET GIVING AMOUNT.
            CREATE BILL (AMOUNT = AMOUNT, ALL OF DELIVERED-ORDER) END
OVER
                                  COMMENT: COMMODITY-SET is also used as a
                                           domain elsewhere.
END-OF-DESCRIPTION
```

6-3. CONCLUSION

This case study has a fairly complicated timing requirement and data selection. It also requires a lot of calculative description due to the need to decode forms. It has very special requirement in terms of timings.

The description shows that it is possible to describe all the necessary aspects of a real life information system without making use of comments or host language procedures. Such a statement can be used for the analysis of data requirements etc.

-

CHAPTER 7

CONCLUSIONS

Information system design for administrative systems has always taken enormous time and man-power. Most often, the reason for this is a communication gap between the user and the system analyst, created by an inadequate description of the user's problem. This calls for many iterations during various phases of the system development cycle. The problem of reducing the time and efforts required for the information system development has been taken up in this thesis. The main idea is to introduce the computer as a tool, in the system development cycle.

A lot of work is being done in this direction. Systems like ISDOS and HRCTC.SYSTEM-1BOTTOM PART etc., are some of the systems, which have tried to achieve this, by taking up the automation of the design phase as the main task. Automation of problem acquisition phase is being tried in two different ways : one is that of describing the functions of the organisation, and the other is that of using languages with tuple-oriented data selection.

In this thesis a high level language is developed for describing information processing systems. This language has features for describing the inputs, outputs, data transformations and system timings with the help of data oriented control structures. It is shown that business processing systems can be completely described in this language

without the help of procedures written in host languages. All the features of this language are based on two underlying concepts.

1. User should be allowed to specifically describe all the interactions of the information processing system, with his organisation. This information could be such as update timings or instructionsfor picking up line numbers from a document to be used as product-codes.

2. Business systems are concerned with classes of subsets of data. Mechanisms of data selection and generation for such subsets are essential for high level system descriptions. This fact has not been recognized in the existing literature.

It is shown that algorithmic analysis of the kind, done by other system like ISDOS, can be done on system descriptions given in the language described in this thesis. The data needs can be identified and can be matched with the data availability. It is also shown that real life systems analysis is much more than mechanically checking for the presence of required data items.

It is established that the block-structure of SDL can be exploited to algorithmically analyse the system description to generate questions about the system for doing the following:

1. To guide the user to give more correct descriptions of the system.
2. To get design information from the user related to data cardinality and integrity.

It is felt that to analyse and design information systems, it is important to understand the purpose of data transformations. It has been seen that in the model building approach, the descriptions become a large set of small concepts. Putting the pieces together, without a mechanisms for structuring, is very difficult. Further work is to be done to work out better user interfaces. But they will be useful only when data oriented structural descriptions and not property oriented descriptions are given. New concepts to enrich the user interface vocabulary may be developed. The concepts should be abstracted in a manner usable for data processing. Here we have developed the subset concept. Similarly an abstraction which could identify real entities is very much needed. In this thesis, time has been abstracted to a usable level. Similarly, a generalised concept for 'measure' could be developed. These abstractions could make the level of the description less clerical and more managerial.

It will also be very useful if a generator could be developed which may not use very efficient design strategies but may atleast create one feasible design and generate the desired information processing system.

REFERENCES

[ACK 71] Ackoff, Russell, L., "Toward a System of System Concepts," Management Sciences, Vol. 17, No. 11, July 1971, pp. 661-671.

[ADR 72] ADR International Corp., Applied Data Research Inc., Route 208 Center, Princeton, New Jersey, U.S.A. This firm has developed: SAM (Systems Analysis Machine), Meta COBOL, Librarian, ROSCOE.

[CHM 73] Chamberlin, D.D., and Boyce, R.F., "Using a Structured English Query Language as a Data Definition Facility", RJ 1318, IBM Research Lab., San Jose, California, December 1973.

[COD 71] Codd, E.F., "A Data Base Sublanguage Foundedoon the Relational Calculus", RJ893, IBM Res. Lab., Yorktwon Heights, N.Y., July 1971

[COU 73] Couger, D.D., "Evolution of Business System Analysis Techniques," ACM Computing Surveys, Vol. 5, No. 3, September 1973, pp. 167-198.

[COU 74] Couger, J.D. (Ed.); and Knapp, R.W., "System Analysis Techniques", John Wiley & Sons, New York, 1974.

[CDA 59] CODASYL, "Conference on Data System Languages", ACM, 1133 Avenue of the Americas, New York, 10036, 1959.

[CHA 71] Chadler, E.W., Nador, P., "Technique for Identification of User's Information Requirements", IFIP-71, North Holland Publ.

[CHA 74] Chadler, E.W., Mazzawi, A.N., "Towards Integrated MIS," IFIP-74, Bell, Northern Electric Research Ltd., North Holland.

[CLI 71] Clifton, H.D., "Order Control and Sales Accounting"Data Processing System Design, Business Book Limited, 1971, pp. 21-42.

[DOY 76] Doyle, Patrick, "Every Object is a System - A Cognitive Approach," Published by P. Doyle, Duncannon, New Ross, Ireland, 1976

[DAT 74] Date, C.J., Codd, E.F., "The Relational and Network Approaches : Comparison of the Application Programming Interfaces," RJ1401, IBM TJ WRC, N.Y., June 6, 1974.

[DEU 74] Denssen, P., "Description of Processes", International Computing Symposium, North Holland Publ., 1974, pp. 11-17.

[FEH 73] Fehder, P.L., "Representations Independent Language - Part I Introduction and Subsetting Operations", RJ1121, IBM TJWRC, N.Y., "Representation Independent Language - Part II - Derivations and Insert", RJ 1251, IBMTJWRC, N.Y., July 1973.

[FOS 76] Fosdick, L.D., and Osterweil, L.J., "Data Flow Analysis in Software Reliability," ACM Computing Surveys, Vol. 8, No. 3, September 1976, pp. 305-330.

[GER 76] Garritsen, R., "A Preliminary System for the Design of DBTG Data Structures", CACM, October 1975, p. 551.

[GOL 75] Goldberg, P.C., "Automatic Programming", RC5148, IBMTJWRC, N.Y., September 1974.

[HAM 75] Hamner, M.M.; Howa, W.G., Wladawsky, I., "An Interactive Business Definition System," Comp. Sc. Dept., IBMTJWRC, Yorktown Heights, N.Y., 10598, 1975.

[HER 74] Hershey, E.A., "PSA - An Introduction," ISDOS Project, Department of Industrial Engineering and Operation Research, University of Michigan, 1974.

[FRY 76] Fry, J.P.; and Sibley, E.H., "Evolution of Data Base Management System", ACM Computing Surveys, Vol. 8, No. 1, March 1976, pp. 7-42.

[IA 1-62] Information Algebra, Phase I Report, CACM, Vol. 5, No. 4, April 1962, pp.190-204.

[KRU 75] Kruksal, B.; Howe, W.G., "Preliminary Definition of Documents, Business Definition Language," RC 5204, IBM TJWRC, N.Y., January 1975.

[KUR 74] Kruksal, B.; Howe, W.G., "Formal Definition of Documents, Business Definition Language," RC5191, IBMTJWRC, N.Y., December 1974.

[LAN 63] Langefors, B., "Some Approaches to the Theory of Information, Systems," BIT3, 1963, pp. 229-259.

[LOM 60] L. Lombardi, "Theory of Files," Proceedings of EJCC, 1960.

[LAN 65] Langefors, B., "Information System Design Computation Using Generalised Matrix Algebra," BIT 52, 1965.

[LAN 63] Langefors, B., "Some Approaches to the Theory of Information Systems," BIT 34, 1963.

[LYN 69] Lynch, H.J., "Accurately Defined Systems - A Technique in System Documentation," Data Base, Vol. 1, No. 1, 1969, pp. 6-18.

[LUN 75] Lundberg, M., 1."Information Analysis - a Systeneering," 2."Object System Studies - Systeneering," Technical Reports, Royal Institute of Technology, University of Stockholm, Sweden, 1975.

[MYE 63] Myers, D.H., "A Time Grid Techniques for the Design of Information Systems," IBM System Research Institute, N.Y., 1963.

[MAR 73] Martin, W.A.; Krunland, R.B.; Sunguroff, A., "MCRE MAPL : Specifications and Basic Structures," Automatic Programming Group, Project MAC, MIT, Internal Memo 8, February 1973.

[NCR 70] "A Study Guide for ADS - Accurately Defined Systems," EP9522-01, NCR, Dayton, Ohio, 1970.

[NUN 71] Numamaker, J.F., "A Methodology for the Design and Optimisation of Information Processing Systems," AFIPS, Vol. 38, 1971, pp. 283-293.

[NUN 74] Numamaker, J.F., Jr., "ADS - A Problem Statement Language," CACM, December 1974, p. 674.

[NIS 73] Nissen, H.E., et. al., '1.'Formalised Description of Object Systems - Tool in Development of Information System," '2.'Techniques for Description of Objective System," '3.'Information Analysis - A Method in Design of Information Systems," TRITA-DBADB - 4204, 1972-04-25, Royal Institute of Technology, Sweden, 1973.

[OSM 75] Osman, I.M., "The Partitioning of Data Base into Subfiles Matching User Queries," Lecture Notes of Computer Science, 26 Springer-Verlag series.

[PRY 75] Prywes, N.S., "Automatic Generation of Computer Programs," Technical Report PEN 19174, Univ. of Pennsylvania, September 1975.

[RHO 72] Rhodes, J., "Beyond Programming : Practical Step and Towards the Automation of Data Processing System Creation Systems Analysis Techniques," John Wiley & Sons, September 1972, pp. 328-335.

[RUT 75] Ruth, G.R., "Status of Proto System I : Bottom Part," Internal Memo 16, Automatic Programming Group, Project MAC, MIT, January 1975.

[SEN 72] Senko, M.E., Altman, E.B., Astrahan, M.M., Fehder, P.L., Wang, C.P., "A Data Independent Architecture Model 1 : Four Levels of Description from Logical Structures to Physical Search Structures," RJ982, IBM Res., February 1972.

[SEN 75] Senko, M.E., "The DDL in the Control of a Multilevel Structured Description DIAM II with FORAL," Maths. Science Department, IBMTJWRC, York Heights, New York, 1975, pp. 239-257.

[TEI 72] Teichrcow, D., "A Survey of Languages for Stating Requirements for Computer Based Information Systems," FJCC 1972, pp. 1203-1224.

[TEI 71] Teichrow, D., Sayani, H., "Automation of System Building," Datamation, August 15, 1971, pp. 25-30.

[TEI 67] Teichreow, D., "ISDOS - A Research Project to Develop Methods for Automatic Design and Construction of Information Processing Systems," ISDOS Working Paper 1, Case Institute of Technology, August 1967.

[TEI 71] Teichroew, D., "Problem Statement Analysis : Requirements for Problem Statement Analysis," ISDOS Working Paper No. 43, Department of Industrial Engineering, University of Michigan, Ann Arbor, 1971, pp. 20-53.

[ZLO 74] Zloof, M.M., "Query by Example," RC4917, IBM TJWRC, New York, July 1974.

[ZLO 75] Zloof, M.M., Jong, "System for Business Automation," RC5302, IBM, TJWRC, March 1975.

[ZLO 77] Zloof, M.M., "System for Business Automation - Query by Example," CACM, June 1977.

APPENDIX I

```
100300  GRAMMER DEFINITION.
100500  GRAMMER-ID. REPORT.
100700  OPTIONS ARE LIST NODES XREF COMPILE.
100900  DEFINE SIZE TEXT = 80 BYTES.
101100  DEFINE SIZE ENTITY = 30 BYTES.
101330  DEFINE SIZE STACK = 9 ENTIES.
101500  PROCEDURE LIST.
101700            DATA-NEED.
101900             REMOVE-DI,
102100            ERROR-ROUTINE 1 PARAMETERS,
102300            ERROR-TERMINATE,
102500            VALUE-OF-DI,
102700            PLACE-OF-DI,
102900            MATCH-DI,
103100            SAVE-VALUE,
103300            SAVE-POSITION?
103500            INCREASE-LEVEL,
103700            RESET-POSITION,
103900            ENTER-DI,
104100            ENTER-DI-GENERATED,
104300            ENTER-FILE-NAME,
104500            DECREASE-LEVEL .
104700  GRAPH SECTION.
104900  SET-AUTOREAD-LINE(1).
105100  BIG-QUARY: "REPORT"  GO TO DATA-DESCRIPTION
105300                       ELSE ERROR-ROUTINE(1)
105500                            ERROR-TERMINATE.
105700  END-OF-QUERY:             DATA-NEED.
105900   %  EASY TO USE  SAVE-DI INSTEAD OF ENTITY STACK.
106100  PLACE-AND-VALUE:       CLEAR-ENTITIES.
106300  REPEAT-LINE:                INTERROGATE-STACK.
106500      "OVER" OR *END*   SET-AUTOREAD-LINE(0), GO TO END-OF-QUERY.
106700         "VALUE"      ELSE  GO TO POSITION.
106900         *TOGGLE*     ELSE ERROR-ROUTINE(5) ; GO TO REPEAT-LINE.
107100         *LITERAL* OR *INTEGER*    SAVE-VALUE , RETURN-ENTITY,
107300                           VALUE-OF-DI, SAVE-ENTITY,
107500                            GET-ENTITY ; GO TO REPEAT-LINE
107700                         ELSE ERROR-ROUTINE(3); GO TO REPEAT-LINE.
107900  POSITION:
108100          "FROM"      ELSE  GO TO LINE.
108300          *TOGGLE*    ELSE ERROR-ROUTINE(6) ; GO TO REPEAT-LINE.
108500          *INTEGER*       SAVE-POSITION, RETURN-ENTITY,
108700                          PLACE-OF-DI, SAVE-ENTITY
108900                             GET-ENTITY, GO TO REPEAT-LINE,
109100                         ELSE   ERROR-ROUTINE(4) ; GO TO REPEAT-LINE.
109300  LINE:
109500         *TOGGLE*         RETURN-ENTITY,
109700                          ERROR-ROUTINE(7).
109900        *WORD*            MATCH-DI,
110100                       ELSE  GO TO  LINE-END.
110300          *SW11*          SAVE-ENTITY,
110500                                GO TO REPEAT-LINE,
110700                          ELSE  SUPPRESS-ENTITY.
110900  LINE-END: *ANY-WORD*    ERROR-ROUTINE(8).
```

```
111100                                  GO TO REPEAT-LINE.
111300  DATA-DESCRIPTION:                   INCREASE-LEVEL
111500                                      GO TO ANY-TYPE.
111700  ANY-TYPE: "BEGIN"                   SUPPRESS-ENTITY, GO TO NEXT-LEVEL.
111900            "END"                     GO TO END-DATA-DESC.
112100          "OVER" OR *END*               GO TO END-DATA-DESC    .
112300            "FORMAT"                  GO TO PLACE-AND-VALUE
112500                          ELSE GO TO NEXT-NODE.
112700  REPORT-MEDIUM-ACTION:
112900                  "NEW-LINE"              RESET-POSITION,
113100                                        GO TO ANY-TYPE.
113300                  "NEW-PAGE"              RESET-POSITION,
113500                                        GO TO ANY-TYPE.
113700  SET-FUNCTION-DI:
113900                   "COUNT"
114100                                    ELSE GO TO SET-SELECTION-ACTION.
114300                   *WORD*                 ENTER-DI,
114500                                    ELSE  ERROR-ROUTINE(9),
114700                                          GO TO ANY-TYPE.
114900                   "AS"                   GO TO NEXT-NODE,
115100                                    ELSE ERROR-ROUTINE(10)
115300                                          GO TO ANY-TYPE.
115500          *WORD*                   ENTER-DI-GENERATED,
115700                                    GO TO ANY-TYPE,
115900                             ELSE ERROR-ROUTINE(10).
116100                           GO TO ANY-TYPE.
116300  SET-SELECTION-ACTION:
116500                   "FOR"                GO TO NEXT-NODE
116700                                      ELSE GO TO DATA-STRING.
116900  SUB-SET-SELECT : "EACH"   GO TO NEXT-NODE ,
117100                                      ELSE GO TO GIVEN-VALUE-SELECT.
117300                    "OF"              ELSE GO TO UNQUE-VALUE-SUBSET.
117500                     *WORD*             ENTER-FILE-NAME
117700                                        GO TO NEXT-LEVEL
117900                                ELSE    ERROR-ROUTINE(11)
118100                                        GO TO ANY-TYPE.
118300  UNIQUE-VALUE-SUBSET:
118500                        *WORD*          ENTER-DI, GO TO NEXT-LEVEL
118700                              ELSE      ERROR-ROUTINE(11)
118900                                        GO TO ANY-TYPE.
119100  GIVEN-VALUE-SELECT:
119500                        *WORD*
119500                                        ENTER-DI
119700                                  ELSE ERROR-ROUTINE(11)
119900                                        GO TO ANY-TYPE.
120100                "="       ELSE    GO TO IGNORE-FOR.
120300                          *LITERAL*     GO TO NEXT-LEVEL.
120500  IGNORE-FOR:                             REMOVE-DI,
120700                                          ERROR-ROUTINE(11),
120900                                          GO TO ANY-TYPE.
121100  DATA-STRING: *WORD*          ENTER-DI, GO TO ANY-TYPE,
121300                       ELSE GO TO NEXT-NODE.
121500           ","      GO TO ANY-TYPE.
121700            *ANY-WORD*  ERROR-ROUTINE(12)     GO TO ANY-TYPE.
```

```
121900 NEXT-LEVEL: "BEGIN"                          GO TO LABEL-1,
122100                          ELSE                ERROR-ROUTINE(13),
122300                                              GO TO LABEL-1,
122500 END-DATA-DESC:                             DECREASE-LEVEL, EXIT(1).
122700 LABEL-1:                                   PERFORM DATA-DESCRIPTION (1),
122900                                            GO TO ANY-TYPE.
123100 PROGRAM SECTION.
123300 IDENTIFICATION DIVISION.
123500 PROGRAM-ID.  BIGQRY
123700 ENVIRONMENT DIVISION.
123900 CONFIGURATION SECTION.
124100 INPUT-OUTPUT SECTION.
124300 FILE-CONTROL.
124500     SELECT REPSPC ASSIGN  TO READER.
124700     SELECT DTNEED ASSIGN  TO PRINTER.
124900 DATA DIVISION.
125100 FILE SECTION.
125300 FD  REPSPC.
125500 01  REC-IN          PIC  X(80).
125700 FD  DTNEED.
125900 01  OUT-REC.
126100  03 FILLER          PIC  X.
126300  03 AREA1           PIC  X(32).
126500  03 AREA2           PIC  X(33).
126700  03 AREA3           PIC X(33).
126900  04 AREA4              X(33).
127100 WORKING-STORAGE  SECTION.
127300 77  I                PIC 99.
127500 77 J                 PIC 99.
127700 77 K           PIC 99.
127900 77 L           PIC 99.
128100 77 SAVE-VAL         PIC  X(30).
128300 77 SAVE-POS         PIC  999.
128500 77 DEEPEST-LEVEL          PIC 99  VALUE  ZERO.
128700 77 FIRST-DEEPEST          PIC 99  VALUE ZERO.
128900 77 LAST-DEEPEST           PIC 99  VALUE ZERO.
129100 77 REL-LV                 PIC 99  VALUE ZERO.
129300 77 REL-NO                 PIC 99  VALUE ZERO.
129500 77 REL-DI-SN              PIC 99  VALUE ZERO.
129700 77 WORK-1               PIC  X(30).
129900 77 TEXT                 PIC  X(80).
130100 77 TEXT-OUT    PIC      X(132).
130300 01 DATA-TABLE.
130500    02 D-TABLE      OCCURS    100   TIMES.
130700       03 D-SN                        PIC   99.
130900       03 D-NAME                      PIC   X(30).
131100       03 D-LEVEL                     PIC   99.
131300       03 D-N-POS                     PIC   999.
131500       03 D-N-POS-DI                  PIC   99.
131700       03 D-VAL                       PIC   X(30).
131900       03 D-TYPE                      PIC   X.
132100 01  CURRENT.
132300    02  C-SN                          PIC   99.
132500    02  C-NAME                        PIC   X(30).
```

```
132700        02 C-LEVEL                    PIC  99.
132900        02  C-NEXT-POS               PIC 999.
133100        02  C-NEXT-POS-AFT-DI-NO     PIC  99.
133300 01     ER-MESSAGE.
133500  02    ER-1.
133700   03       FILLER  PIC  X(20) VALUE  "NOT A REPORT".
133900   03       FILLER  PIC  X(20) VALUE "END BY DEFAULT".
134100   03       FILLER  PIC  X(20)  VALUE "VALUE-INCORRECT".
134300   03       FILLER  PIC  X(20) VALUE "VALUE IGNORED".
134700   03       FILLER  PIC  X(20) VALUE "POSITION IGNORED".
134900   03       FILLER  PIC  X(20) VALUE "FORMAT IGNORED".
135100   03       FILLER  PIC  X(20) VALUE "WORD IGNORED".
135300   03       FILLER  PIC  X(20) VALUE "COUNT IGNORED".
135500   03       FILLER  PIC  X(20) VALUE "AS IGNORED".
135700   03       FILLER  PIC  X(20) VALUE "FOR IGNORED".
135900   03       FILLER  PIC  X(20) VALUE "NOT-RECOGNISED".
136100   03       FILLER  PIC  X(20) VALUE "BEGIN-ASSUMED".
136300   03       FILLER  PIC  X(20)  VALUE "                ".
136500  02 ER-2    REDEFINES  ER-1.
136700  03 ER-3    OCCURS 14   TIMES   PIC  X(20).
136900  PROCEDURE DIVISION.
137100  USER SECTION.
137300  OPEN-FILES.
137500        OPEN INPUT REPSPC OUTPUT DTNEED.
137700      MOVE  "REPORT-ANALYSIS" TO TEXT-OUT.
137900      PERFORM WRITE-LINE THRU USER-EXIT.
138100      GO TO USER-EXIT.
138300  READ-LINE.   IF TEXT NOT = "OVER" THEN
138500      READ REPSPC  INTO TEXT AT END MOVE "OVER" TO TEXT.
138700      GO TO USER-EXIT.
138900  WRITE-LINE.
139100*
139300      WRITE OUT-REC FROM TEXT-OUT AFTER 2. MOVE SPACES TO TEXT-OUT.
139500       MOVE SPACES TO OUT-REC.
139700  DUM-1.
139900      GO TO USER-EXIT.
140100  ERROR-ROUTINE.
140300      MOVE PARAMETER-1 TO I.
140500      MOVE  ER-3(I) TO AREA1.
140700      MOVE ENTITY TO AREA2.
140900      PERFORM WRITE-LINE-1.
141100      GO TO USER-EXIT.
141300  ERROR-TERMINATE.
141500      GO TO CLOSE-PARA.
141700  MATCH-DI.
141900      MOVE 1 TO I. MOVE ZERO TO SW11 J.
142100  MATCH-DI-1.
142300      IF D-NAME(I) = ENTITY MOVE 1 TO SW11 MOVE I TO J
142500         GO TO MATCH-DI-EXIT.
142700      IF I = C-SN  GO TO MATCH-DI-EXIT.
142900      ADD 1 to I.  GO TO MATCH-DI-I.
143100  MATCH-DI-EXIT.
143300  EXIT.
```

```
143500 MATCH-DI-EXIT-1.
143700     GO TO USER-EXIT.
143900 WRITE-LINE-1.
144100      WRITE OUT-REC AFTER 2.
144300       MOVE SPACES TO OUT-REC.
144500 ENTER-DI.
144700     ADD 1 TO C-SN.
144900     MOVE ENTITY TO D-NAME(C-SN), C-NAME.
145100     MOVE C-SN TO D-SN(C-SN).
145300     MOVE C-LEVEL TO D-LEVEL(C-SN).
145500     MOVE C-NEXT-POS TO D-N-POS(C-SN).
145700      MOVE C-NEXT-POS-AFT-DI-NO TO D-N-POS-DI(C-SN).
145900      MOVE C-SN TO C-NEXT-POS-AFT-DI-NO.
146100      MOVE "Y" TO D-TYPE(C-SN).
146300 ENTER-DI-EXIT.
146500     GO TO USER-EXIT.
146700 SAVE-VALUE.
146900       MOVE ENTITY-VALUE TO SAVE-VAL.
147100     GO TO USER-EXIT.
147300 SAVE-POSITION.
147500     MOVE  INTEGER      TO SAVE-POS.
147700     GO TO USER-EXIT.
147900 RESET-POSITION.
148100     MOVE 1 TO C-NEXT-POS.
148300     MOVE ZERO TO C-NEXT-POS-AFT-DI-NO.
148500     GO TO USER-EXIT.
148700 ENTER-DI-GENERATED.
148900     PERFORM ENTER-DI.
149100     MOVE "GO" TO D-TYPE(C-SN).
149300     GO TO USER-EXIT.
149500 ENTER-FILE-NAME.
149700     ADD 1 TO C-SN.
149900     MOVE ENTITY TO D-NAME(C-SN).
150100     MOVE "F" TO D-TYPE(C-SN).
150300     MOVE C-LEVEL TO D-LEVEL(C-SN).
150500     GO TO USER-EXIT.
150700 VALUE-OF-DI.
150900     PERFORM MATCH-DI THRU MATCH-DI-EXIT.
151100     MOVE SAVE-VAL TO D-VAL(J). MOVE "C" TO D-TYPE(J).
151300     GO TO USER-EXIT.
151500 PLACE-OF-DI.
151700     PERFORM MATCH-DI THRU MATCH-DI-EXIT.
151900     MOVE SAVE-POS TO D-N-POS(J).
152100     MOVE ZERO TO D-N-POS-DI(J).
152300 REMOVE-DI.
152700     SUBTRACT 1 FROM C-SN.
152900     MOVE D-NAME(C-SN) TO C-NAME.
153100     MOVE C-SN TO C-NEXT-POS-AFT-DI-NO.
153500     GO TO USER-EXIT.
153700 INCREASE-LEVEL.
153900     ADD 1 TO C-LEVEL.
154100     GO TO USER-EXIT.
```

```
154300 DECREASE-LEVEL.
154500     SUBTRACT 1 FROM C-LEVEL.
154700     GO TO USER-EXIT.
154900 DATA-NEED.
155100     MOVE "DATA-NEEDS :" TO AREA1
155300     PERFORM WRITE-LINE-1.
155500 PARA-1.
155700     PERFORM DEEPEST THRU DEEPEST-END.
155900     IF DEEPEST-LEVEL = 1 GO TO PARA-2 ELSE GO TO PARA-1.
156100 DEEPEST.
156300     MOVE 1 TO I.
156500     MOVE ZERO TO DEEPEST-LEVEL.
156700 CHECK-DEEPEST.
156900     IF SW05 = 3
157100     MOVE D-TABLE(I) TO OUT-REC PERFORM WRITE-LINE-1.
157300     IF D-TYPE(I) = "G" OR "C" MOVE ZERO TO D-LEVEL (I).
157500     IF D-LEVEL(I) > DEEPEST-LEVEL THEN
157700         MOVE D-LEVEL(I) TO DEEPEST-LEVEL
157900     MOVE I TO FIRST-DEEPEST.
158100     IF D-LEVEL(I) = DEEPEST-LEVEL THEN
158300     MOVE I TO LAST-DEEPEST.
158500 CHECK-1.
158700     ADD 1 to I.
158900     IF I NOT > C-SN GO TO CHECK-DEEPEST.
159100 REPEAT-DEEPEST.
159300     PERFORM ONCE-DEEPEST THRU ONCE-DEEPEST-END.
159500     IF DEEPEST-LEVEL = 1 GO TO DEEPEST-END.
159700 AGAIN.
159900     SUBTRACT 1 FROM LAST-DEEPEST.
160100     IF D-LEVEL(LAST-DEEPEST) = DEEPEST-LEVEL GO TO AGAIN.
160300 MORE-DEEPEST.
160500     SUBTRACT 1 FROM LAST-DEEPEST.
160700     IF LAST-DEEPEST   FIRST-DEEPEST GO TO DEEPEST-END.
160900     IF D-LEVEL(LAST-DEEPEST) = DEEPEST-LEVEL  THEN
161100     GO TO REPEAT-DEEPEST.
161300     GO TO MORE-DEEPEST.
161500 DEEPEST-END.
161700     EXIT.
161900 ONCE-DEEPEST.
162100      MOVE DEEPEST-LEVEL TO REL-LV.
162300      MOVE LAST-DEEPEST TO I.
162500      MOVE "RELATION" TO AREA1.
162700      ADD 1 TO REL-NO. MOVE REL-NO TO AREA2.
162900      PERFORM WRITE-LINE-1. MOVE ZERO TO REL-DI-SN.
163100        IF DEEPEST-LEVEL = 1 ALD 1 TO I
163300          GO TO PARA-12.
163500* FOR LEVEL 1 A ALL DOMAINS ARE KEY DOMAINS.
163700     MOVE "ATTRIBUTE-DOMAINS" TO AREA1.  PERFORM WRITE-LINE-1.
163900 PARA-10.
164100     ADD 1 TO REL-DI-SN.
164300     MOVE REL-DI-SN TO AREA1.  MOVE D-NAME(I) TO AREA2.
164500     IF D-TYPE(I) = "F" MOVE "IDENTIFICATION-NO" TO AREA3.
164700     IF D-LEVEL(I) = DEEPEST-LEVEL MOVE ZERO TO D-LEVEL(I).
164900       PERFORM WRITE-LINE-1.
```

```
165100 PARA-11.
165300 SUBTRACT 1 FROM I.
165350 IF D-LEVEL(I) = REL-LV GO TO PARA-10.
165400 IF I  -0--
165500     IF D-LEVEL(I) = .0  ,  GO TO PARA-11.
165700     SUBTRACT 1 FROM REL-LV.
165900 PARA-12.  IF REL-LV = ZERO GO TO PARA-14.
166100     MOVE "KEY-DOMAINS" TO AREA1. PERFORM WRITE-LINE-1.
166300 PARA-13.
166500     IF REL-LV = ZERO GO TO PARA-14.
166700 PARA-15.
166900     IF D-LEVEL(I) = REL-LV GO TO PARA-16.
167100     SUBTRACT 1 FROM I.
167300     IF D-LEVEL(I)   REL-LV MOVE "ERROR" TO AREA1 PERFORM
167500     WRITE-LINE-1 GO TO CLOSE-PARA.  GO TO PARA-15.
167700 PARA-16.
167900       PERFORM PARA-10 THRU PARA-11.
168100       GO TO PARA-13.
168300 PARA-14.
168500       EXIT.
168700 ONCE-DEEPEST-END.
168900       EXIT.
169100 PARA-2.
169300       EXIT.
169500*
169700*
169900 CLOSE-PARA.
170100     MOVE "END-OF-JOB" TO AREA1.
170300     PERFORM WRITE-LINE-1.
170500       CLOSE REPSPC  DTNEED.
170700     STOP RUN.
170900 END-OF-JOB.
```